TECHNICKÁ UNIVERZITA V LIBERCI
Fakulta mechatroniky, informatiky
a mezioborových studií

SYSTÉM PRO CENTRÁLNÍ SPRÁVU IDENTIT

Diplomová práce

Studijní program: N2612 – Elektrotechnika a informatika
Studijní obor: 1802T007 – Informační technologie

Autor práce: **Bc. Václav Bohata**
Vedoucí práce: Mgr. Jiří Vraný, Ph.D.

Liberec 2015

TECHNICAL UNIVERSITY OF LIBEREC
Faculty of Mechatronics, Informatics
and Interdisciplinary Studies

CENTRALIZED IDENTITY MANAGEMENT SYSTEM

Diploma thesis

Study programme: N2612 – Electrical Engineering and Informatics
Study branch: 1802T007 – Information Technology

Author: **Bc. Václav Bohata**
Supervisor: Mgr. Jiří Vraný, Ph.D.

Liberec 2015

TECHNICKÁ UNIVERZITA V LIBERCI
Fakulta mechatroniky, informatiky a mezioborových studií
Akademický rok: **2014/2015**

ZADÁNÍ DIPLOMOVÉ PRÁCE

(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

Jméno a příjmení:	**Bc. Václav Bohata**
Osobní číslo:	**M12000197**
Studijní program:	**N2612 Elektrotechnika a informatika**
Studijní obor:	**Informační technologie**
Název tématu:	**Systém pro centrální správu identit**
Zadávající katedra:	**Ústav nových technologií a aplikované informatiky**

Zásady pro vypracování:

1. Seznamte se s aktuálně používanými systémy pro správu identit (identity management). Zejména se zaměřte na problematiku automatizace základních procesů správy identit a problematiku synchronizace identifikačních dat.
2. Získané teoretické znalosti použijte pro rozšíření stávající aplikace pro správu uživatelských účtů, vytvořené v rámci bakalářské práce, o možnosti vytvářet automatické akce pro obvyklé úlohy správy. Dále o možnost synchronizace dat mezi systémy a o serverovou část s rozhraním pro vzdálený přístup k vybraným úlohám správy identit.
3. Návrh prakticky implementujte a otestujte.

Rozsah grafických prací: **dle potřeby**

Rozsah pracovní zprávy: **60 stran**

Forma zpracování diplomové práce: **tištěná/elektronická**

Seznam odborné literatury:

[1] ELISA BERTINO, Kenji Takahashi. Identity Management Concepts, Technologies, and Systems. Norwood: Artech House, 2010. ISBN 16-080-7040-9.
[2] WILLIAMSON, Graham, David YIP, Ilan SHARONI a Kent SPAULDING. Identity management: a primer. Lewisville, TX: Mc Press, 2009. ISBN 15-834-7093-X.

Vedoucí diplomové práce: **Mgr. Jiří Vraný, Ph.D.**
Ústav nových technologií a aplikované informatiky

Datum zadání diplomové práce: **20. října 2014**

Termín odevzdání diplomové práce: **15. května 2015**

prof. Ing. Václav Kopecký, CSc.
děkan

L.S.

prof. Dr. Ing. Jiří Maryška, CSc.
vedoucí ústavu

V Liberci dne 20. října 2014

Prohlášení

Byl jsem seznámen s tím, že na mou diplomovou práci se plně vztahuje zákon č. 121/2000 Sb., o právu autorském, zejména § 60 – školní dílo.

Beru na vědomí, že Technická univerzita v Liberci (TUL) nezasahuje do mých autorských práv užitím mé diplomové práce pro vnitřní potřebu TUL.

Užiji-li diplomovou práci nebo poskytnu-li licenci k jejímu využití, jsem si vědom povinnosti informovat o této skutečnosti TUL; v tomto případě má TUL právo ode mne požadovat úhradu nákladů, které vynaložila na vytvoření díla, až do jejich skutečné výše.

Diplomovou práci jsem vypracoval samostatně s použitím uvedené literatury a na základě konzultací s vedoucím mé diplomové práce a konzultantem.

Současně čestně prohlašuji, že tištěná verze práce se shoduje s elektronickou verzí, vloženou do IS STAG.

Datum:

Podpis:

Poděkování

Touto cestou bych rád vyjádřil poděkování vedoucímu mé diplomové práce Mgr. Jiřímu Vranému, Ph.D. za cenné rady při tvorbě práce.

Abstrakt

Diplomová práce se zabývá návrhem a tvorbou aplikace pro centrální správu identit. Úvodní část se věnuje základním pojmům z oblasti podnikových Identity Management systémů. Představuje existující bezplatná softwarová řešení. Zaměřuje se na jejich základní principy a způsob automatizace správy uživatelů. Porovnává jejich implementace a uživatelská rozhraní.

Hlavní část práce se věnuje návrhu aplikace pro centrální správu uživatelů. Aplikace je určená pro menší organizace (školy, menší firmy, …). Skládá se z několika částí. Jádro všech tvoří serverová aplikace nabízející API pro připojení ostatních částí. Je to hlavní výkonná část celku. Další části jsou tvořeny konektorem, konzolovou aplikací pro správu identit a uživatelským rozhraním pro koncové uživatele. S pomocí konektorů aplikace spravuje koncové databáze uživatelských účtů. Mapuje koncové uživatele spravovaných databází na identity – centrální účty v databázi aplikace. Mapování se provádí automaticky. Identita obsahuje sadu atributů. Aplikace synchronizuje vybrané atributy identity s atributy koncových účtů ve spravovaný systémech. Dokáže kontrolovat obsah dat přenášených z aplikace do koncového systému a naopak a provádět jejich modifikaci. Umí vytvářet, upravovat nebo odstraňovat identity a příslušné namapované koncové účty.

Závěrečná část práce se zabývá implementací navržené aplikace. Popisuje použité technologie a softwarové nebo hardwarové požadavky aplikace. Představuje vytvořený software, jeho strukturu a způsob jeho ovládání.

Klíčová slova

Identity Management, správa uživatelských účtů, RBAC, synchronizace uživatelských atributů, konektor

Abstract

The master thesis describes the design and implementation of centralized identity management software. The first part of the thesis introduces the basic concepts of enterprise identity management systems and describes existing free of charge software solutions, their basic concepts and user management automation. It also compares their implementations and user interfaces.

The main part of the thesis describes the design of centralized identity management software. The application is designed for smaller organizations (schools, smaller companies, …). It consists of four parts. The core part is the server application with an API for the connection of the other parts. It is the main part of the whole application. The next parts are the connector, the user administration interface and the end-user web interface. The connectors manages connected systems (databases) of user accounts. The application maps user accounts of managed remote systems to some identities – user accounts stored in the application's database. Each system account can be mapped to an appropriate identity. The application maps each account automatically. Each identity has some set of attributes. The application synchronizes some identity attributes with attributes of managed account. The content of attributes transmitted to or from the application can be validated and modified. The application can create, modify or delete identities and mapped user accounts.

The last part of the thesis is focused to the implementation of the application. First it describes used technologies and software or hardware requirements of the created application. It also describes the details of the created software, its structure and its use.

Keywords

Identity Management, management of user accounts, RBAC, synchronization of user attributes, connector

Obsah

Přílohy

Seznam obrázků

Seznam tabulek

Seznam syntaxí

Seznam příkazů

Seznam zkratek

API	Application Programming Interface, rozhraní pro programování aplikací
CLI	Command-Line Interface, rozhraní příkazového řádku
GUI	Graphical User Interface, grafické uživatelské rozhraní
HR	Human Resources, lidské zdroje
HTTP	Hypertext Transfer Protocol, protokol pro výměnu hypertextových dokumentů
HTTPS	Hypertext Transfer Protocol Secure, šifrovaný HTTP
IBAC	Identity-based access control, řízení přístupu na základě identity uživatele
IdM	Identity Management, mechanismus správy uživatelských účtů
IEEE	Institute of Electrical and Electronics Engineers
JEXL	Java Expression Language , skriptovací jazyk pro Javu
JPBL	jBPM Process Definition Language, jazyk pro popis procesů
JSON	JavaScript Object Notation, způsob zápisu dat určených k přenosu
JSON-RPC	RPC rozhraní využívající k přenosu zpráv JSON
LDAP	Lightweight Directory Access Protocol, protokol pro přístup k adresářovému serveru
ORM	Object-relational mapping, objektově relační zobrazení
PHP	Hypertext Preprocessor, skriptovací programovací jazyk
RBAC	Role-based access control, způsob řízení oprávnění na základě rolí
RPC	Remote procedure call, vzdálené volání procedur
REST	Representational state transfer, datově orientovaná architektura pro přístup k datům (zdrojům)
SOAP	Simple Object Access Protocol, protokol pro výměnu zpráv
SSL	Secure Socket Layer, vrstva poskytující šifrování a autentizaci dat

SSO	Single Sign-On, řízení přístupu k aplikacím na základě jednotného přihlášení
URL	Uniform Resource Locator, způsob jednoznačného určení zdroje v síti

1 Úvod

1.1 Úvod do problematiky

Správa uživatelských účtů je běžná činnost při administraci informačních systémů. Do většiny systémů se obvykle uživatelé autentizují svým heslem. Jedná se o jeden z nejjednodušších způsobů prokazování své identity, kdy jsou lidé nuceni pamatovat si jedno nebo více hesel, která mohou někdy zapomenout. Poté je na správci systému vytvořit uživatelům hesla nová. Každá změna uživatelských údajů či vytváření nebo mazání účtu vyžaduje určitý administrativní zásah do informačního systému, respektive do databáze uživatelských účtů.

Jestliže je uživatel založen v několika technicky nepropojených, přesto logicky závislých databázích, musí správce provést změnu v každé z nich. Typickým příkladem může být firma, kde zaměstnanci mají účty ve dvou nepropojených systémech, každý s vlastní databází účtů. V každé databázi se nachází telefonní číslo na danou osobu. Změna telefonního čísla musí být provedena samostatně ve všech databázích. Když administrátor upraví uživatelský údaj pouze v jedné databázi, pak dojde k nekonzistenci s druhou databází. Řešením problému může být společná databáze pro všechny připojené systémy. Ne vždy je to možné nebo vhodné řešení. Potom je dobré použít nástroj, který umí vybrané údaje ze všech potřebných databází synchronizovat automaticky.

1.2 Cíl práce

Práce se zabývá tvorbou aplikace pro synchronizaci uživatelských atributů, jejich monitorování a centrální správu. Uživatelský účet (identita) vytvořený v centrální databázi je namapován na konkrétní účty ve spravovaných databázích a na základě nastavení provádí aplikace automaticky jejich synchronizaci. Při změně hodnot atributů může systém na základě nastavení vhodně reagovat – například poslat e-mail uživateli nebo správci při překročení určité hodnoty apod.

Správa uživatelů se provádí přes příkazový řádek tak, aby práce s uživatelskou databází byla co nejefektivnější. Dovoluje administrátorům integrovat příkazy pro správu účtů do vlastních skriptů. Většina akcí umožňuje spuštění nakonfigurovaných schvalovacích

procesů, kde před skutečným zásahem do databáze požádá aplikace o schválení nastavené uživatele.

Koncovým uživatelům je k dispozici webové rozhraní, ve kterém si mohou změnit hesla v propojených databázích a zpracovávat případné schvalovací požadavky.

2 Systémy pro správu identit

2.1 Základní principy a pojmy

Správa identit (identity management) je vlastně správa uživatelů přistupujících k určitým aplikacím. Typicky je identity management systém (IdM) podnikový software, který dokáže z jednoho místa řídit životní cyklus identit [1]. Po nástupu do zaměstnání jsou zaměstnanci zřízeny účty a nějaká oprávnění. Časem získává další přístupy nebo nějaké ztrácí a po rozvázání pracovního poměru jsou mu účty zablokovány nebo smazány.

Identitou ve smyslu identity management systému se rozumí sada atributů, které jednoznačně identifikují osobu či spíše ji rozlišují od ostatních. Podle [2] je identita standardně definována jako kombinace obecných atributů typu jméno, adresa a kontaktní údaje s atributy významnými pro organizaci. Atributy identity by měly být zadávány pouze jednou (centrálně), aby se předešlo chybám při manuálním zadávání do několika systémů.

IdM bývá napojen na autoritativní zdroj – na personální systém, tzv. Human Resources (HR). Odtud provádí IdM tzv. provisioning do spravovaných systémů. Provisioning je proces, při němž se naplňují databáze spravovaných systémů daty o uživatelích. Procesu, při kterém se na základě dat v koncovém systému aktualizují identity v IdM (mapování, aktualizace atributů, …) se říká rekonciliace.

Synchronizace uživatelů, provisioning a případně další operace bývají podpořeny tzv. workflow. Díky tomu může být zajištěna značná automatizace a robustnost procesů v organizaci. Podle [2] musí workflow dodržovat schvalovací procesy v organizaci, kdy různí manažeři schvalují jednotlivé uživatelovy přístupy do firemních systémů. Schvalovací postupy musejí počítat i s případnou nedostupností samotných manažerů. Pokud je manažer na dovolené, pak se schvalování deleguje na někoho jiného. Celý workflow musí být auditovatelný a auditor by měl být schopný zkontrolovat, zda byla dodržena firemní politika.

Kvůli přehlednosti by měla být uživatelská oprávnění přiřazována rolím, ne jednotlivým identitám. Zásadní je správný návrh rolí vč. jejich počtu. Role jsou svázány

s pracovními pozicemi. Metoda přiřazování oprávnění identitám se nazývá identity-based access control (IBAC), metoda přiřazování rolím Role-based access control (RBAC). V organizacích je RBAC efektivnější. Uvědomuje si, že zaměstnanci na pozicích se střídají rychleji, než se mění pracovní povinnosti na dané pozici [3]. Při změně přeřazení zaměstnance na jinou pozici stačí upravit přidělené role, není zapotřebí přiřazovat jednotlivá oprávnění.

2.2 Typická architektura používaných IdM

IdM různých výrobců se od sebe sice liší, ale jejich architektura je velice podobná. Identity management systémy fungují na principu dodání potřebných dat do koncových systémů [1]. Jejich architektura je centrálně orientovaná. Koncové systémy jsou k IdM připojeny skrze tzv. konektory. Konektory jsou poté součástí IdM, abstrahují spojení s koncovými systémy a mohou na nich provádět různé operace. Koncovým systémem je cokoliv s vlastní databází uživatelů, atributy v systému jsou mapovány na atributy IdM. Synchronizace/provisioning účtů se provádí automaticky či jako naplánovaná úloha. Komunikace s koncovým systémem je závislá na konektoru, může využívat standardní protokoly (LDAP, …). Kromě základních funkcí poskytují dnešní IdM i Single Sign-On (SSO) řešení apod. Těm se tato práce nebude věnovat.

2.3 Přehled vybraných IdM

S nadhledem lze říci, že každá větší softwarová společnost vyvíjí vlastní komerční IdM s širokou funkcionalitou vhodný k nasazení i do velkých korporací. Mezi těmito společnostmi jsou například IBM, Oracle, Microsoft, Dell nebo Hitachi. Na trhu se však nacházejí i některé bezplatné produkty. Vzhledem ke komplikovanosti identity management systémů je vhodné i tyto systémy svěřit do správy zkušeným odborníkům a platit si technickou podporu. Na základě [4] o vybraných bezplatných IdM pojednává následující text. Všechny níže zmíněné produkty jsou napsány v jazyce Java.

2.3.1 CzechIdM

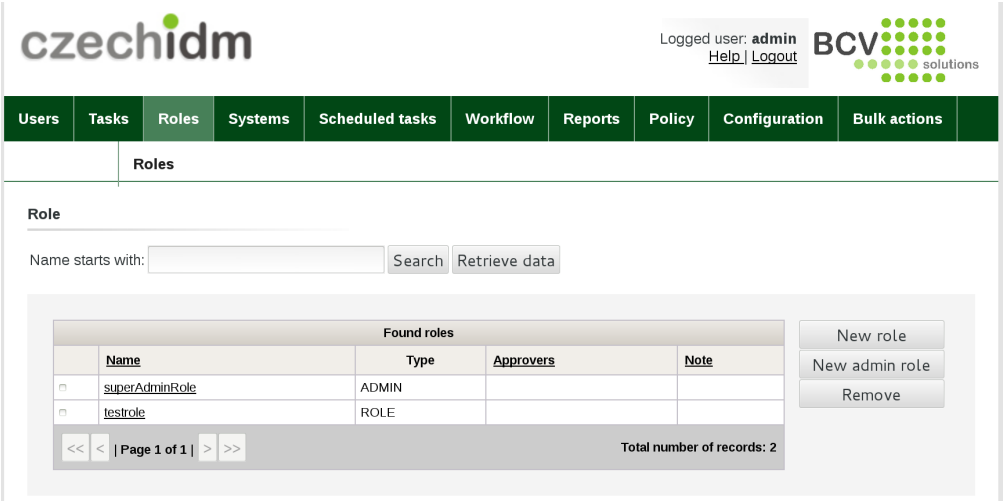
Tento český IdM systém je vyvíjen společností BCV solutions s.r.o. V polovině roku 2014 bylo [5] oznámeno zpřístupnění softwaru pod svobodnou licencí LGPL. Mnoho jeho principů je inspirováno Sun IdM. Běží na aplikačním serveru JBoss. Podporuje

jednoduché role, organizační struktury a provisioning rolí (přiřazení systémů k rolím). Role mohou být v jednoduché hierarchii a mít předdefinované hodnoty atributů – buď staticky nebo dynamicky pomocí BeanShell kódu.

Flexibilita CzechIdM je do značné míry dosažena používáním pravidel. Pravidla jsou BeanShell skripty prováděné v příslušných částech aplikace. Pravidla mohou být mnoha druhů. Například transformační, korelační či notifikační. Transformační pravidla najdou uplatnění při práci s daty koncových systémů. Ukázkovým pravidlem může být kód pro negaci hodnoty atributu při načítání z koncového systému do databáze IdM a naopak. Korelační pravidla slouží pro mapování identit při rekonciliaci či synchronizaci koncového systému s CzechIdM. Notifikační pravidla se spouštějí při přiřazování rolí a slouží pro informování (např. e-mailem) určitých osob o provedené operaci. Existuje celá řada dalších pravidel, která jsou popsána v [6].

V CzechIdM řeší různá workflow velkou část aplikační logiky. Běží na enginu jBPM a jsou psaná v jazyku JPDL (jBPM Process Definition Language). Workflow je vlastně stavový automat, může odpovídat konkrétnímu firemnímu procesu. Například při odchodu zaměstnance jsou některé účty v koncových systémech smazány, jiné zablokovány nebo převedeny na někoho jiného [7].

Uživatelské rozhraní aplikace je jednoduché a strohé, ale vyskytují se v něm chyby. Celkový dojem kazí problém s opětovným načítáním stránek po provedení změn, kdy je potřeba provést manuální obnovení stránky ve webovém prohlížeči. Ukázka rozhraní je na obrázku 2.1.



Obr. 2.1: Webové rozhraní CzechIdM – seznam rolí

CzechIdM používá relativně zastaralý software a celkově je náročný na operační paměť. V závislosti na HW může jeho spouštění trvat několik minut. Při testování v linuxové distribuci CentOS 7 se ukázalo, že ke spuštění CzechIdM a optimální reakční době webového rozhraní je zapotřebí alespoň 2 GB operační paměti.

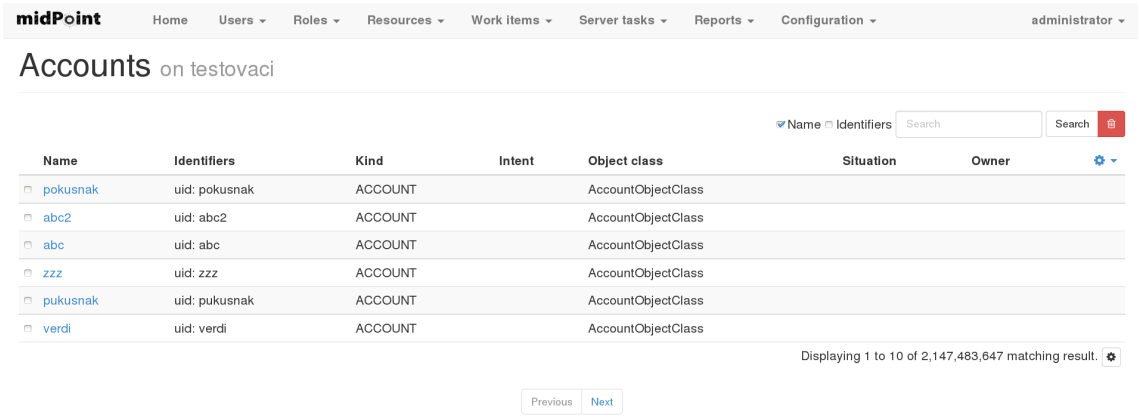
2.3.2 Evolveum midPoint

Slovenský software midPoint je moderní identity management systém vyvíjený pod licencí Apache License 2.0. Používá aplikační server Apache Tomcat. Je připraven pro různé druhy nasazení bez nutnosti psaní a testování funkcí, které se běžně vyskytují při zavádění IdM softwaru. Má výbornou podporu RBAC a umožňuje flexibilní provisioning rolí. Role lze přiřazovat dočasně a mohou být uspořádány do hierarchie. Nemají pouze statické nastavení, jejich parametry lze upravovat individuálně během přidělování [8]. Díky tomu lze redukovat počet rolí v IdM. Typickým parametrem může být budova či oddělení, kde daný zaměstnanec pracuje. Díky tomu lze na základě definovaných výrazů nastavovat hodnoty atributů daného účtu. Evolveum midPoint podporuje modelování téměř každé organizační struktury. Ta je úzce spojena s RBAC, workflow procesy a autorizačním mechanismem.

Jako workflow engine se používá Activity. Workflow obvykle nemusí být nijak přizpůsobovány. U obvyklého schvalovacího workflow stačí nastavit schvalovatele.

Nová verze midPoint přináší podporu generické synchronizace. Díky tomu umožňuje synchronizovat v podstatě jakýkoliv druh objektu s téměř jakýmkoliv jiným objektem. Toto může být využito k synchronizaci rolí, skupin, organizačních jednotek, projektů aj. Takovou vlastnost má i OpenIDM, nicméně ten na rozdíl od midPoint nerozumí vztahům mezi objekty.

Webové rozhraní pro správu se snaží být ergonomické a mít moderní vzhled. Je funkčně velice bohaté a IdM celkově komplexní, proto není podpora nejnovějších funkcí v GUI vždy aktuální. Ukázka rozhraní je na obrázku 2.2. S aplikací lze komunikovat i pomocí RESTful či SOAP API.



Obr. 2.2: Webové rozhraní midPoint – seznam uživatelů na spravovaném systému

2.3.3 OpenIAM IdM Community Edition

OpenIAM vychází ve dvou verzích: Community Edition a Enterprise Edition. Community edition je bezplatná verze vydaná pod licencí GNU/GPL v3. Integruje identity a access management (řízení přístupu, autentizační politiky, SSO).

Podporuje správu organizačních jednotek, skupin, rolí a zdrojů. Tento koncept je jednoduchý, ale komplexní. Nemusí být vždy jasné, kdy kterou část použít. Dokumentace [9] uvádí, že skupiny slouží pro modelování organizační struktury, role pro modelování funkcí v podniku. Uživatel je členem jedné ze skupiny, skupiny i role mohou mít hierarchické uspořádání. Skupiny mohou být přiděleny organizacím, zdroje uživatelům. Zdrojem může být mnoho věcí. Například systém, který má být spravován nebo objekt, který potřebuje být chráněn (URL, …).

OpenIAM obsahuje workflow engine Activity. Stejně jako CzechIdM nebo Syncope používá workflow při každé operaci. Výhodou tohoto přístupu je značná flexibilita, nevýhodou vliv na výkon aplikace.

OpenIAM má vestavěné pouze základní modely pro RBAC, organizační strukturu, rekonciliaci a synchronizaci. Složitější modely vyžadují programování.

Aplikace obsahuje dvě webová uživatelská grafická rozhraní. Administrátorskou konzoli a samoobsluhu (self-service), ve které si mohou uživatelé spravovat svůj profil, resetovat hesla, prohledávat adresář firemních uživatelů aj., jak uvádí [10]. Kromě těchto rozhraní nabízí OpenIAM RESTful a SOAP API.

2.3.4 ForgeRock OpenIDM

Tento generický a velice flexibilní IdM, jehož zdrojový kód je vydán pod licencí CDDL 1.0, nabízí základní identity management funkcionalitu, která může být rozšířena skriptováním. OpenIDM působí spíše jako framework a pro většinu nasazení bude nutné si ho „doprogramovat“ dle konkrétních potřeb.

OpenIDM může teoreticky pracovat s jakýmkoliv druhem objektů, nerozumí však jejich vzájemným vztahům. Evolveum midPoint může být nakonfigurováno tak, že po vytvoření role v IdM se založí nová LDAP skupina díky tomu, že zná vzájemné vztahy mezi typy objektů. Toho lze docílit i u OpenIDM, jen je potřeba to naprogramovat. OpenIDM nemá zabudovanou podporu pro organizační strukturu a obsahuje primitivní podporu RBAC. Obsahuje workflow engine Activity. Ten ale není nedílnou součástí OpenIDM a může být vypnut.

IdM neobsahuje praktické administrátorské rozhraní, GUI je spíše jednoduchý prototyp. Všechno v OpenIDM je flexibilní a programovatelné, vytvoření GUI by bylo velmi náročným úkolem. Aplikace se ovládá skrze REST rozhraní.

2.3.5 Apache Syncope

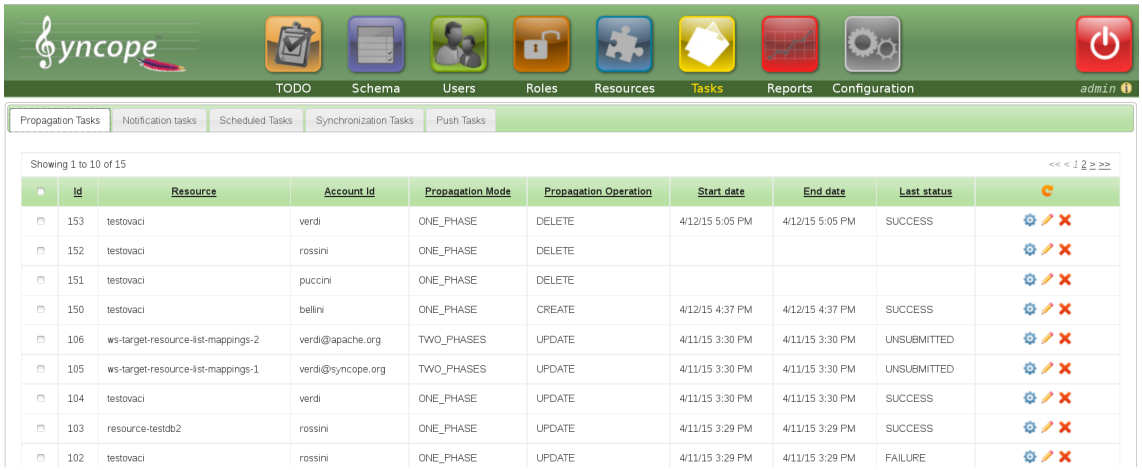
Apache Syncope je software distribuovaný pod licencí Apache License 2.0. Jeho architektura a funkcionalita se podobá Sun IdM. Pro jednoduchá nasazení může být použit tak, jak je. U složitějších nasazení je možné ho rozšířit.

Schéma má plochý model, definuje pojmenované atributy, které mohou mít hodnoty několika primitivních datových typů. Podporuje pouze přímé mapování atributů uživatele/identity na atributy účtu ve spravovaném systému. Lze určit odkud kam se mají data synchronizovat. Nelze ale definovat žádné transformace. Jak uvádí [11], někdy je vhodné, aby hodnota atributu byla kombinací hodnot jiných atributů, například atribut *fullname* jako spojení *firstname* a *lastname*. To umožňují tzv. odvozené atributy, které se definují pomocí JEXL (Java Expression Language). Některé atributy nemusí mít svoje hodnoty uloženy v úložišti Syncope a jsou rovnou čtené přímo z připojeného systému. Takový atribut nazývá Syncope virtuální. Nevýhodou virtuálních atributů je nedostupnost jejich hodnot v případě nedostupnosti připojeného systému.

Provisioning je řízený pomocí workflow. To umožňuje Syncope adaptovat se na mnoho různých situací. Nevýhodou tohoto řešení je složité řešení konzistence. Ta musí být řešena samotným workflow, což může být nespolehlivé. Syncope umožňuje vybrat si používané workflow enginy viz [12].

Syncope podporuje koncept uživatelů, rolí a členství, synchronizaci uživatelů/identit s účty v systémech a rolí se skupinami, nicméně chybí podpora pro organizační struktury.

Aplikace má oddělené webové grafické uživatelské rozhraní od základní části, která provádí provisioning. Komunikují spolu pomocí REST rozhraní. Ukázka webového rozhraní je na obrázku 2.3.



Obr. 2.3: Webové rozhraní Syncope – úlohy

3 Návrh aplikace

V návrhu aplikace se odráží zkušenosti z návrhu a provozu aplikace vytvořené v rámci bakalářské práce [13]. Původní návrh počítal s centrální správou nezávislých databází uživatelských účtů. Účty v koncových systémech byly automaticky propojeny s aplikací na základě uživatelského jména a samostatně spravovány. Uživatelské atributy měly obraz v centrální aplikaci, se kterým byly synchronizovány. Aplikace neuměla automaticky synchronizovat atributy mezi spravovanými systémy. Existovala manuální aktualizace dat, se kterou pomáhal mechanismus proměnných. Zkušenosti z provozu ukázaly, že hromadná správa velkého množství uživatelů ve webovém grafickém rozhraní byla velice efektivní, ale na úkor efektivní správy jednotlivých uživatelů. Mechanismus oprávnění definoval přístupy až na úroveň hodnot atributů. Bylo možné nastavit, že určitá skupina uživatelů má oprávnění zapisovat hodnotu atributu, pokud odpovídá nastaveným pravidlům atd. Podobně se postupovalo při čtení atributů, díky čemuž se musela oprávnění po změnách hodnot atributů automaticky přepočítávat. To bylo velice rychlé a oprávnění umožňovala detailní ladění přístupů administrátorů. Nicméně nejen po přidání spravovaného systému bylo nutné upravit značné množství oprávnění u všech vytvořených skupin.

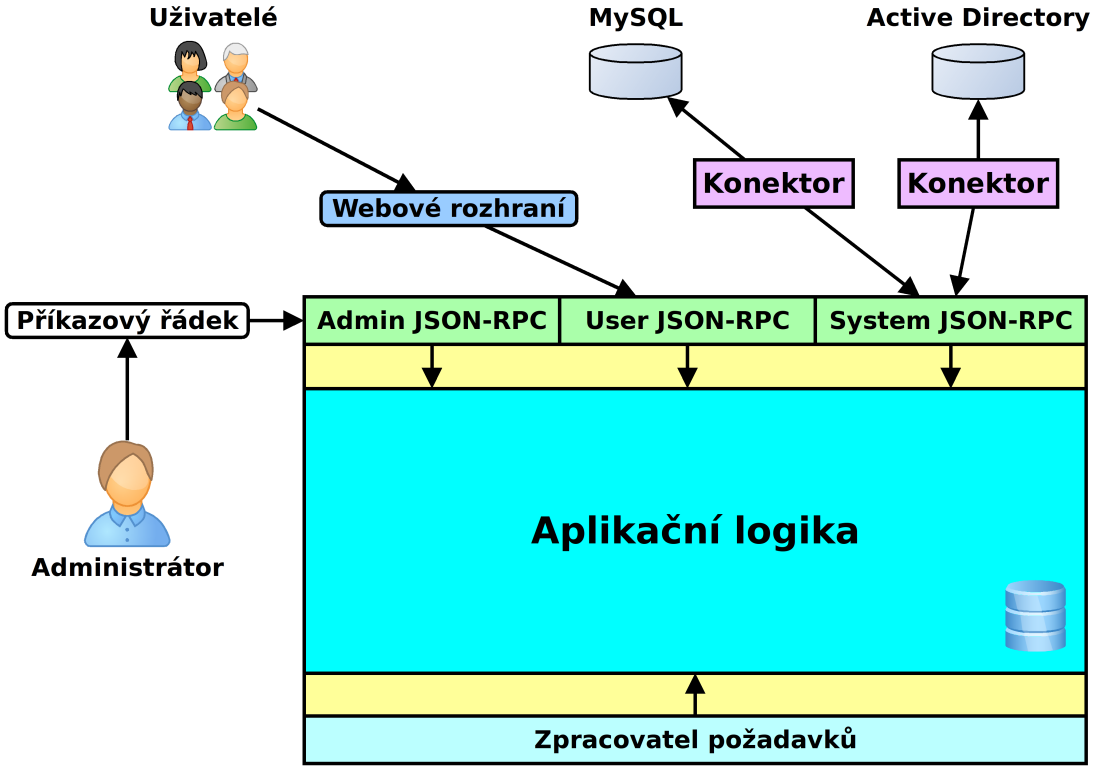
Vzhledem k rozdílným (ačkoliv relativně podobným) principům fungování aplikace s funkcionalitou v bakalářské práci bylo nutné provést nový návrh celého softwaru. Navrhovaná aplikace je cílena na menší organizace, nenabízí běžné vlastnosti podnikových identity management systémů. Neimplementuje workflow, schvalovací procesy jsou součástí oprávnění a mapování koncových účtů probíhá automaticky.

3.1 Architektura aplikace

3.1.1 Popis architektury

Hlavní výkonná část aplikace je umístěna na serveru. Prostřednictvím webového serveru a protokolu HTTP nabízí několik JSON-RPC (RPC využívající JSON pro předávání zpráv) rozhraní pro připojení administrátorů, koncových uživatelů a spravovaných systémů. Na serveru je v pravidelných intervalech spouštěna komponenta na zpracování schvalovacích požadavků.

K aplikaci se připojují administrátoři, koncoví uživatelé a spravované systémy. Administrátoři na svém počítači v příkazovém řádku spravují uživatele, respektive identity v aplikaci. Koncoví uživatelé (včetně administrátorů) mají k dispozici webové uživatelské rozhraní, ve kterém si mohou nastavovat hesla do spravovaných systémů a zpracovávat požadavky čekající na jejich schválení. Hlavní aplikace se nikdy nepřipojuje ke koncovým systémům, koncové systémy se připojují k ní pomocí konektorů. Konektor zná spravovaný systém a na základě potřeby v něm provádí všechny potřebné akce. Na obrázku 3.1 je zobrazena architektura aplikace včetně ukázky dvou připojených konektorů.



Obr. 3.1: Architektura aplikace

3.1.2 Princip správy identit

Administrátor nikdy nespravuje identity v aplikaci přímo. Pouze vytváří požadavky na vytvoření, smazání či úpravu určité identity. O další osud požadavků se stará komponenta na zpracování schvalovacích požadavků – zpracovatel požadavků (dále jen zpracovatel). Součástí požadavku je schvalovací proces. Pokud má schvalovací proces definované nějaké schvalovatele, pak zpracovatel pozastaví zpracování požadavku,

informuje e-mailem schvalovatele (je-li to možné) a čeká na jejich reakce. Ti ve webovém rozhraní pro uživatele potvrdí nebo zamítnou požadavek. Na základě jejich reakcí či po expiraci životnosti požadavku provede zpracovatel smazání nebo potvrzení požadavku a o výsledku operace se pokusí informovat e-mailem jeho autora. Pokud požadavek neobsahuje žádné schvalovatele, provede zpracovatel rovnou potvrzení požadavku.

3.1.3 Rozhraní příkazového řádku

Veškerá správa identit se provádí přes rozhraní pro příkazový řádek. To umožňuje administrátorům rychle a efektivně spravovat menší či větší množství identit. Je navržené tak, aby umožňovalo použití ve skriptech. Kromě přihlašování není interaktivní, všechny potřebné údaje se zadávají jako parametry příkazu. Při přihlašování zadá administrátor své jméno a heslo. Po úspěšné autentizaci vygeneruje výkonná část aplikace přihlašovací token s omezenou životností, který si příkazové rozhraní uloží do lokálního souboru. Poté může administrátor na základě svého oprávnění vytvářet různé požadavky. V případě, že má administrátor antipatie k příkazovému řádku a je zdatný v programování, může si v libovolném jazyce napsat webové rozhraní a připojit ho k aplikaci prostřednictvím JSON-RPC.

3.1.4 Webové rozhraní pro koncové uživatele

Ve webovém rozhraní si mohou po přihlášení koncoví uživatelé standardně prohlížet hodnoty atributů vlastní identity, mají-li k tomu oprávnění, a měnit hesla v koncových systémech. V případě, že jsou uvedeni jako schvalovatelé některých požadavků, pak mohou potvrzovat či zamítat běžící požadavky čekající na jejich schválení. V případě potřeby mohou požádat na určitou dobu o delegování požadavků na někoho jiného či zrušit aktuální delegování.

3.1.5 Konektor

Konektor zastupuje spravovaný systém, respektive slouží jako rozhraní mezi spravovaným systémem a výkonnou (serverovou) částí aplikace. Je to aktivní program, který se pravidelně připojuje k serveru a na základě určeného algoritmu provádí požadované úkony ve spravovaném systému a posílá data serverové aplikaci.

3.2 Logická struktura objektů

3.2.1 Schémata atributů

Každá identita může mít přiřazeno určité množství atributů, každý s teoreticky libovolným počtem unikátních hodnot. Atributy mají předem definované schéma. Některé hodnoty ve schématech atributů nelze upravovat po jejich vytvoření. Definice každého atributu obsahuje:

- datový typ (nelze později upravovat),
- nastavení šifrování (nelze později upravovat),
- název atributu,
- název skupiny,
- popis atributu,
- výchozí hodnoty,
- validátor,
- filtr.

Datovým typem se určuje způsob nakládání s atributem: omezení hodnot dané konkrétním datovým typem, způsob ukládání v databázi, jaké operace mohou být použity při vyhledávání podle jeho hodnot (menší než, rovná se, …) a způsob porovnávání hodnot, které obsahuje. Aplikace rozeznává následující datové typy:

- BINARY: binární data obecného významu, maximálně $300 * 2^{20}$ bajtů, porovnávání rozlišuje velká a malá písmena, vyhledávání podle hodnot nemožné,
- STRING: řetězcové hodnoty, maximálně $200 * 2^{20}$ bajtů, porovnávání rozlišuje velká a malá písmena, vyhledávání podle „odpovídá hodnotě“ nebo „neodpovídá hodnotě“ (lze použít * odpovídající libovolnému počtu znaků a ? odpovídající jednomu libovolnému znaku),
- CISTRING: viz STRING, porovnávání nerozlišuje velká a malá písmena,

- INTEGER: celočíselný datový typ, 32bitové či 64bitové hodnoty v závislosti na platformě, vyhledávání s použitím klasických binárních operátorů,
- FLOAT: číslo s plovoucí řádovou čárkou, přesnost v závislosti na platformě, typicky IEEE 754, vyhledávání s použitím klasických binárních operátorů,
- BOOLEAN: logický datový typ, vyhledávání stejně jako u číselných datových typů.

Nastavení šifrování je přepínač určující, zda jsou hodnoty atributů před uložením do databáze transparentně šifrovány klíčem, který byl automaticky vygenerován při prvním použití šifrování v aplikaci. Šifrování zvyšuje bezpečnost tím, že případnému útočníkovi nestačí ukrást pouze databázi aplikace, ale musí získat i klíč uložený v textovém souboru. Pokud je atribut šifrovaný, nelze ho použít ve vyhledávacích podmínkách při hledání identit.

Název atributu jednoznačně identifikuje atribut v rámci celé aplikace. Název skupiny slouží ke zvýšení přehlednosti při čtení atributů a k jejich logickému uspořádání, stále však platí, že název atributu musí být unikátní. Kromě těchto identifikačních údajů obsahuje schéma popis. Ten by měl jednoznačně charakterizovat atributy tak, aby koncový uživatel jednoduše rozeznal, k čemu se jednotlivé atributy používají.

Někdy se dají hodnoty atributů předpokládat, proto může být užitečné jejich automatické doplňování. Každý atribut obsahuje prázdnou či neprázdnou množinu výchozích hodnot. Ty se použijí tehdy, je-li identitě přiřazován atribut bez uvedení jeho hodnot.

Validátor a filtr jsou objekty, které obsahují pravidla pro kontrolu, respektive filtraci hodnot ukládaných do atributů přidělených identitě. Validačními a filtračními pravidly se detailně zabývá kapitola 3.3.

3.2.2 Systém

Konektor, který se připojuje k aplikaci pracuje s podmnožinou namapovaných atributů. Ty kromě jiného určuje objekt systém. Obsahuje následující položky:

- název systému,
- popis,

- klíč vyžadovaný při autentizaci konektoru,
- určení schopnosti pracovat s hesly,
- formát, ve kterém se automaticky generují nová hesla,
- nastavení mapování uživatelů na identity vč. nastavení odstraňování účtů,
- filtr pro transformaci uživatelského jména nového účtu,
- nastavení mapovaných atributů.

Název jednoznačně identifikuje systém v rámci aplikace. Popis systému by měl být výstižný natolik, aby koncoví uživatelé mimo jiné pochopili, kam si nastavují svá hesla. Při připojování konektorů vyžaduje aplikace úspěšnou autentizaci nakonfigurovaným klíčem.

Ne každý systém musí podporovat práci s hesly. S tím aplikace počítá a umožňuje nastavit, že systém nebude manipulovat s žádnými hesly. To se hodí v případech, kdy se uživatelé do spravovaného systému autentizují jiným způsobem, například certifikátem.

Jestliže je systém nakonfigurovaný s podporou hesel, pak musí aplikace nějakým způsobem získat heslo potřebné při přidávání identit do koncového systému (respektive vytváření účtů a mapování na identity aplikace). V takovém případě má dvě možnosti: použít heslo identity nebo vygenerovat nové. Pokud koncový systém podporuje formát hesel (délka, složitost, …) používaný aplikací, může aplikace dodat stejné heslo do spravovaného systému. To je možné díky tomu, že hesla identit jsou uložená v čitelné podobě, respektive transparentně zašifrována stejně, jako v případě atributů. Jestliže to možné není, pak lze nastavit formát automaticky generovaného hesla. Pokud je nějaký formát nastaven, pak na jeho základě provede aplikace vygenerování náhodného nového hesla pro každý přidaný účet nezávisle na to, jaké heslo má nastavené identita. Obsah nových hesel určuje řetězec, jehož formát je znázorněn syntaxí 3.1.

```
znaky[ minimální_počet[ maximální_počet]]&znaky...
```

Syntaxe 3.1: Syntaxe generovaných hesel

Tabulka 3.1 obsahuje seznam příkladů různě nastavených formátů hesel.

Tab. 3.1: Příklady formátů hesel

Formát	Význam
`abcdef`	Heslo obsahuje právě jeden znak: a, b, c, d, e nebo f.
`abcdef&xyz`	Heslo obsahuje právě dva znaky, prvním je a, b, c, d, e nebo f, druhým x, y nebo z.
`ab 2xy 1 2&abc\&\\`	První dva znaky vygenerovaného hesla budou a nebo b. Další jeden nebo dva znaky budou x nebo y. Posledním znakem hesla bude a, b, c, & nebo \.
`čřžýXx()._ 8 16`	Heslo bude obsahovat 8 až 16 znaků. Tyto znaky se budou skládat ze znaků č, ř, ž, ý, X, x, (,), . nebo _.

Aplikace potřebuje znát, jakým způsobem má mapovat identity na existující účty v koncových systémech a jak se má chovat v případě, kdy v koncovém systému chybí potřebný účet. Automatické mapování se řídí rovností zvolených hodnot. Pokud všechny zvolené hodnoty identity odpovídají hodnotám v koncovém systému, pak aplikace provede namapování účtu na odpovídající identitu. Těmito hodnotami mohou být uživatelské jméno (respektive název identity) a libovolné namapované atributy. Pokud v koncovém systému chybí požadovaný účet, pak se ho aplikace pokusí vytvořit pouze za předpokladu, že není nastaveno ignorování tohoto stavu. Uživatelské jméno účtu je název identity po projití transformačním filtrem. Po smazání identity vydá aplikace po nakonfigurovaném čase pokyn pro smazání všech příslušných namapovaných účtů v koncových systémech. Jak na tento pokyn bude koncový systém reagovat záleží na konektoru.

Konektor může pracovat pouze s těmi atributy, které jsou v objektu systém nakonfigurované jako mapované. Každé nastavené mapování obsahuje:

- mapovaný atribut viz 3.2.1,
- určení, zda se má zahrnout pro porovnávání hodnot při automatickém mapování identit,
- nastavení směrů synchronizace, tzn. jestli synchronizovat z koncového systému do aplikace a naopak,
- nastavení chování při první synchronizaci hodnot (nahradit hodnoty v koncovém systému hodnotami identity či naopak nebo sloučení hodnot),

- filtry, které nastavují transformační pravidla (mezi čtenou hodnotou atributu koncového systému a zápisem atributu identity a naopak).

V tabulce 3.2 je uveden příklad nastaveného mapování atributů. Atribut *mail* nikdy nebude aktualizován ze strany připojeného systému, proto na nastavení chování při první synchronizaci nezáleží. Při mapování identity na existující účet ve spravovaném systému musejí hodnoty atributů *firstName* a *lastName* účtu ve spravovaném systému odpovídat hodnotám atributů identity. U atributů *mail*, *firstName* a *fullName* se při transformaci hodnoty ze systému do aplikace a naopak provádí ořez prázdných znaků. Detailní popis filtračních pravidel je v kapitole 3.3.

Tab. 3.2: Příklad konfigurace mapování atributů

Atribut	Kontrola rovnosti při mapování	Synchronizace		Chování při první synchronizaci	Pravidla ve filtru	
		Z aplikace do koncového systému	Z koncového systému do aplikace		Z aplikace do koncového systému	Z koncového systému do aplikace
mail	NE	ANO	NE	sloučit hodnoty	trim	trim
firstName	ANO	ANO	ANO	nahradit hodnoty v systému	trim	trim
lastName	ANO	ANO	ANO	nahradit hodnoty v systému	trim	trim
loginsCnt	NE	NE	ANO	nahradit hodnoty v aplikaci	-	-

3.2.3 Role

Veškeré zdroje získává identita prostřednictvím rolí. Objekt role obsahuje:

- název role,
- popis,
- seznam atributů,
- seznam systémů,
- prioritu,
- seznam oprávnění.

Název role jednoznačně identifikuje roli, popis slouží k lepší orientaci administrátorů při přidělování či odebírání rolí identitám. Role nemohou obsahovat další role, nelze je vnořovat.

Identita nemůže obsahovat libovolné atributy. Množinu atributů použitelných pro identitu určují role. Po přidělení role získává identita nárok na použití atributů specifikovaných v roli. Role může určit, zda jsou atributy povinné či nepovinné. Role nelze přidělit, nemají-li všechny povinné atributy nastavené hodnoty. Z toho vyplývá, že pokud identita dané hodnoty atributů nemá nastavené díky v minulosti přiděleným rolím, musí se specifikovat při přiřazování role. Pokud se nespecifikují, zkusí aplikace použít výchozí hodnoty. Nejsou-li výchozí hodnoty nastavené, pak přidělení role selže. V případě odebrání role jsou identitě vymazány všechny atributy, které nejsou definované žádnou z dalších přidělených rolí.

Jakýkoliv atribut může mít v roli nastavené fixní hodnoty, které nemusejí odpovídat validačním pravidlům ve schématu atributu. Hodnoty atributu posílané koncovému systému se skládají z hodnot nastavených identitě sloučených s fixními hodnotami ve všech identitě přidělených rolích. Výsledný seznam hodnot obsahuje pouze unikátní hodnoty, případné duplicitní hodnoty jsou zahozeny. Fixně stanovené hodnoty nelze použít ve validačních nebo filtračních pravidlech a nejsou součástí identity.

Po získání členství v roli jsou identitě přiřazené nastavené koncové systémy. V závislosti na konfiguraci jednotlivých systémů jsou identitě vytvořeny koncové účty nebo je spárována s účty existujícími.

Aplikace implementuje RBAC model oprávnění. Všechna oprávnění určuje role, identita je získává skrze členství v roli. Oprávnění definuje, ke kterým rolím mají členové zdrojové role přístup a jaké operace mohou na těchto rolích provádět. U některých oprávnění lze nastavit schvalovací procesy. V takovém případě se nejprve kontroluje konkrétní oprávnění, pokud je oprávnění nalezeno, pak se spustí nastavený schvalovací proces. Může se stát, že oprávnění definuje více rolí s různými schvalovacími procesy. Pokud je identita členem takových rolí, pak je použit schvalovací postup role, která má vyšší prioritu (nižší číslo priority). V tabulce 3.3 je seznam možných oprávnění. Některá oprávnění určují přístup ke konkrétním zdrojům v cílové roli, u některých nelze zdroj nastavit.

Tab. 3.3: Seznam nastavitelných oprávnění

Název oprávnění	Zdroj	Schvalovací proces	Popis oprávnění
CREATE_IDENTITY		lze přiřadit	Vytvoření identity s takovými rolemi, ke kterým má zdrojová role oprávnění CREATE_IDENTITY. Každá identita musí být členem alespoň jedné role.
DELETE_IDENTITY		lze přiřadit	Smazání identity, která je členem cílové role.
DELEGATE_APPROVAL		lze přiřadit	Delegovat oprávnění na identitu, která je členem cílové role.
DELETE_DELEGATED_APPROVAL		lze přiřadit	Smazat jakékoliv delegování z mé identity, která je členem cílové role.
CHANGE_NAME		lze přiřadit	Změnit uživatelské jméno identitě, která je členem cílové role.
CHANGE_PASSWORD		lze přiřadit	Změnit heslo identitě, která je členem cílové role.
CHANGE_ENABLED		lze přiřadit	Povolit či zakázat účet identitě, která je členem cílové role.
CHANGE_USERINTERFACEACCESS		lze přiřadit	Změnit nastavení přístupu k uživatelskému rozhraní aplikace identitě, která je členem cílové role.
CHANGE_ADMININTERFACEACCESS		lze přiřadit	Změnit nastavení přístupu k administrátorskému rozhraní aplikace identitě, která je členem cílové role.
MOVE_TO_CONTAINER		lze přiřadit	Přesunout identitu do nějakého kontejneru, pokud je identita členem cílové role.
ADD_ROLE	role	lze přiřadit	Přiřadit identitě roli (specifikována zdrojem), pokud je identita členem cílové role.
REMOVE_ROLE	role	lze přiřadit	Odebrat identitě roli (specifikována zdrojem), pokud je identita členem cílové role.
ALTER_ATTRIBUTE	atribut	lze přiřadit	Upravit identitě atribut (specifikován zdrojem), pokud je identita členem cílové role.
READ_ATTRIBUTE	atribut	nelze přiřadit	Číst atribut (specifikován zdrojem) identity, který je členem cílové role.

Nastavení oprávnění demonstruje příklad v tabulce 3.4. Tabulka popisuje oprávnění, která jsou přiřazená roli *studijni*. Identita *novakova* je členem této role a získává proto následující oprávnění.

Tab. 3.4: Příklad nastaveného oprávnění pro roli studijni

Cílová role	Oprávnění	Zdroj	Schvalovací proces
studenti	ADD_ROLE	stravnici	pridat_stravnika_menzy
studenti	REMOVE_ROLE	stravnici	odebrat_stravnika_menzy
studenti	READ_ATTRIBUTE	zustatekUctu	
studenti	DELETE_IDENTITY		vymazat_studenta
spravci	DELEGATE_APPROVAL		spr_delegace_schvalovani
studijni	DELETE_DELEGATED_APPROVAL		-

Identita/uživatelka *novakova* má oprávnění přidat identity, které jsou členy role *studenti* do role *stravnici* za předpokladu, že je její požadavek schválen ve schvalovacím procesu *pridat_stravnika_menzy*. Rovněž má oprávnění odebrat takové identitě roli *stravnici*, pokud je požadavek schválen v procesu *odebrat_stravnika_menzy*. Členům role *studenti* může prohlížet hodnoty atributu *zustatekUctu* a v případě potřeby může takové identity vymazat po schválení v procesu *vymazat_studenta*. Jestliže *novakova* odjíždí na dovolenou, může požádat o předání svých schvalovacích povinností na kohokoliv, kdo je členem role *spravci* po schválení v *spr_delegace_schvalovani*. Po návratu z dovolené smaže *novakova* delegovaná schvalování bez nutnosti schválit tento krok v nějakém procesu.

3.2.4 Kontejner

Aplikace neobsahuje mechanismus organizačních jednotek. Identity je však možné logicky rozdělovat do vytvořených kontejnerů. Kontejnery mají různá omezení, nelze umístit do libovolného kontejneru libovolnou identitu. Kontejnery nelze vnořovat, každá identita je umístěná právě v jednom kontejneru. Kontejner má následující vlastnosti:

- název, který jednoznačně kontejner identifikuje,
- popis,
- seznam možných rolí.

Do kontejneru lze umístit pouze takovou identitu, jejíž všechny role kontejner povoluje. Lze nastavit, zdali je daná role vyžadována. Pokud identita neobsahuje kontejnerem vyžadovanou roli, pak nelze identitu do kontejneru umístit. Kromě toho umožňuje

kontejner nastavit, zda je daná role výchozí pro identity nově vznikající v tomto kontejneru. Toto nastavení se nevztahuje na existující identity, které se do tohoto kontejneru pouze přesouvají.

Pokud identita nově vznikající v kontejneru neobsahuje všechny povinné nebo výchozí role, jsou jí automaticky přidány. Pokud identita nemá specifikované žádné hodnoty atributů vyžadovaných přiřazenými rolemi, jsou automaticky použity jejich výchozí hodnoty (pokud nějaké existují). Díky tomu lze dosáhnout značného zjednodušení při spravování identit.

3.2.5 Identita

Identita je objekt představující uživatelský účet v aplikaci. Tvoří ho následující hodnoty:

- uživatelské jméno (název identity),
- přihlašovací uživatelské heslo,
- stav identity (povolena/zablokována),
- nastavení přístupů k uživatelskému a administrátorskému rozhraní aplikace,
- kontejner, ve kterém je umístěn,
- seznam přidělených rolí,
- hodnoty atributů.

Uživatelské jméno je unikátní v rámci celé aplikace. Heslo musí mít délku alespoň 8 znaků. Musí obsahovat jedno velké a jedno malé písmeno, jedno číslo a jeden jiný znak. Nesmí se podobat žádné hodnotě uložené v atributech identity (ani jménu identity). Při ukládání hodnot atributů identity jsou provedeny nejprve nastavené filtrační, poté validační pravidla.

3.2.6 Schvalovací proces

Součástí požadavků mohou být schvalovací procesy. Schvalovací proces se skládá ze schvalovacích kroků. Každý krok obsahuje seznam schvalovatelů – identit. Schvalovací proces i schvalovací kroky mají nastavenou životnost. Po překročení této životnosti dojde k automatickému schválení či zamítnutí (podle konfigurace procesu) kroku či celého požadavku, pro který byl proces spuštěn.

Identita může požadavky na určitou dobu delegovat na jinou identitu. V takovém případě je v běžícím procesu ve vhodný čas nahrazena původní identita identitou novou.

Tabulka 3.5 zobrazuje konfiguraci dvou ukázkových schvalovacích procesů.

Tab. 3.5: Ukázkové schvalovací procesy

Proces			**Krok**			
Název	**Automatická akce**		**Pořadí**	**Automatická akce**		**Schvalovatel**
	Typ	**Čas spuštění**		**Typ**	**Čas spuštění**	
vymazat_studenta	schválit	1 hodina	1	schválit	10 minut	petrzel
						janackova
			2	zamítnout	50 minut	lister
odebrat_stravnika_menzy	zamítnout	2 minuty	1	zamítnout	1 hodina	hubert

Po vytvoření požadavku a spuštění procesu *vymazat_studenta* se bude čekat 10 minut na uživatele (identity) *petrzel* nebo *janackova*. Jestliže mají uživatelé nastavený e-mail, budou aplikací najednou kontaktováni. Pokud někdo z nich požadavek schválí, pak se bude pokračovat následujícím krokem procesu. Jestliže někdo z nich požadavek zamítne, pak se proces ukončí a požadavek bude odmítnut. Pokud uživatelé nebudou do 10 minut reagovat, krok schvalovacího procesu požadavku se schválí a bude se pokračovat druhým krokem. V druhém kroku je na uživateli *lister* schválení nebo zamítnutí požadavku. Pokud nebude do 50 minut reagovat, celý požadavek bude odmítnut.

Po vytvoření požadavku a spuštění procesu *odebrat_stravnika_menzy* se bude čekat 2 minuty na reakci schvalovatele *hubert*. Pokud *hubert* do 2 minut požadavek neschválí, pak bude odmítnut.

O výsledku všech požadavků bude aplikace informovat jejich tvůrce za předpokladu, že mají nastavený e-mail.

3.3 Kontrola a transformace dat

Na různých místech v aplikaci dochází ke kontrole a transformaci dat. Kontrola se provádí tzv. validátory. Validátor obsahuje validační řetězec a popis. Nejčastěji se používá pro kontrolu hodnot atributů při ukládání identit nebo jeho samostatný validační řetězec jako součást filtru. Filtr obsahuje filtrační pravidla, vykonání každého pravidla je podmíněno validačním řetězcem. Pomocí filtrů je v aplikaci zajištěna transformace dat. Používá se při ukládání atributů identity, při přenášení hodnoty atributů systému do atributů identit a naopak nebo pro transformaci názvů identit do uživatelských jmen koncových účtů během vytváření účtů na koncových systémech.

3.3.1 Validační řetězce

Validační řetězec je specificky formátovaný text obsahující validační pravidla. Dle typu může validační pravidlo obsahovat několik argumentů. Formát validačního řetězce znázorňuje syntaxe 3.2.

```
pravidlo[ & pravidlo...][ | pravidlo[ & pravidlo...]]...
```

Syntaxe 3.2: Syntaxe validačního řetězce

Pokud je potřeba, lze části filtračního řetězce uzavírat do závorek. Aplikace obsahuje několik základních pravidel viz tabulku 3.6. Seznam pravidel lze kdykoliv rozšířit. Pravidla se vztahují k atributům a primárně kontrolují hodnoty atributu, pro který jsou definované. Každé pravidlo dostane seznam všech hodnot daného atributu a vrací pravdu nebo nepravdu (true nebo false).

Tab. 3.6: Validační pravidla

Pravidlo	Použití	Popis
email	email	Kontroluje, zda všechny hodnoty atributu představují platnou e-mailovou adresu.
ip	ip	Kontroluje, zda všechny hodnoty atributu představují platnou IP adresu.
ipv4	ipv4	Kontroluje, zda všechny hodnoty atributu představují platnou IPv4 adresu.
ipv6	ipv6	Kontroluje, zda všechny hodnoty atributu představují platnou ipv6 adresu.
numerical	numerical	Kontroluje, zda jsou všechny hodnoty atributu číselné.

Pravidlo	Použití	Popis
range	range MIN MAX	Kontroluje, zda se všechny hodnoty atributu nacházejí mezi čísly MIN a MAX.
strlen	strlen MIN MAX	Kontroluje, zda je délka řetězců u všech hodnot atributu alespoň MIN a maximálně MAX.
regexp	regexp VYRAZ	Kontroluje, zda všechny hodnoty atributu odpovídají Perl kompatibilnímu regulárnímu výrazu VYRAZ.
inlist	inlist HODN1 HODN2 …	Kontroluje, zda se každá hodnota atributu nachází v seznamu hodnot HODN1 HODN2 … Nezávisle na datovém typu se seznam prohledává s ohledem na velikost písmen.
valuescount	valuescount MIN MAX	Kontroluje, zda je počet hodnot alespoň MIN a maximálně MAX.
not	not PRAVIDLO ARG	Vrací pravdu, pokud vnořené PRAVIDLO použité s argumenty ARG vrací nepravdu.
any	any	Vrací vždy pravdu.
getvar	getvar PROM PRAVIDLO ARG	Kontroluje, zda hodnota proměnné PROM vyhovuje pravidlu PRAVIDLO, kde ARG jsou argumenty použité pro PRAVIDLO. PROM je proměnná uložená v databázi, vztahuje se ke konkrétní identitě.
contains	contains PRAVIDLO ARG	Kontroluje, zda alespoň jedna z hodnot vyhovuje pravidlo PRAVIDLO. ARG jsou argumenty pro PRAVIDLO.
look	look ATRIBUT PRAVIDLO ARG	Kontroluje, zda hodnoty atributu ATRIBUT vyhovují pravidlu PRAVIDLO, kde ARG jsou argumenty pro PRAVIDLO.

Validační pravidla jsou svázána s atributy. Vykonávají se samostatně pro každý kontrolovaný atribut, ale mají k dispozici hodnoty všech ostatních atributů konkrétní prověřované identity (případně koncového účtu) a speciální virtuální atribut obsahující hodnotu, která představuje název identity. Argumenty pravidel nemusejí být statické hodnoty, můžou se odkazovat na jakoukoliv hodnotu jakéhokoliv dostupného atributu pomocí speciálních proměnných:

- `%$%username%`, která přestavuje název identity
- `%`*`atribut`*`%`*`N`*`%`, která přestavuje *N*-tou hodnotu atributu *atribut*.

Někdy může být užitečné použít v argumentech pravidel znaky, které mají ve validačním řetězci speciální použití: (, &, |, _ aj. Takové znaky lze escapovat znakem \. Jestliže mají být součástí argumentů mezery, lze využít znak _. Tento znak nahradí aplikace při použití mezerou.

Příklad použití

Následující příklad demonstruje použití validačních řetězců při kontrole hodnot atributů. Tabulka 3.7 obsahuje seznam atributů s validačními řetězci, které jsou součástí validátoru ve schématu atributů a s konkrétními hodnotami použitými u nějaké identity. Zobrazuje výsledek kontroly hodnot vzhledem k použitým validačním řetězcům. Ve sloupci s kontrolovanými hodnotami je zobrazeno u každé hodnoty její pořadí tak, jak je hodnota uložena v databázi. Pokud atribut neobsahuje žádné hodnoty, pak se posuzuje tak, jako by ho identita neobsahovala (lze, pokud není v přiřazené roli jako povinný).

Tab. 3.7: Použití validačních pravidel na konkrétní hodnoty

Název atributu	Validační řetězec	Kontrolované hodnoty	Výstup validátoru
mail	valuescount 1 1 & strlen 1 2500 & email	1. jan@email.com 2. novak@firma.cz	nepravda – není splněna první podmínka (počet hodnot je více než jedna)
fullName	valuescount 1 1 & strlen 1 50	1. Kim-Chun Papaduos Jan Oliver-Radim Theo Raduanovič Novák	nepravda – není splněna druhá podmínka (délka řetězce je větší než 50)
category	valuescount 1 1 & inlist internista externista	1. internista	pravda – jsou splněny všechny podmínky
group	valuescount 1 4 & (inlist administrativa manageri | regexp /^skup[0-9]+$/)	1. administrativa 2. skup5 3. manageri	pravda – jsou splněny všechny podmínky (počet hodnot je od 1 do 4 a hodnota je v seznamu administrativa, manageri nebo začíná řetězcem skup a pokračuje nějakým číslem)
primaryGroup	valuescount 1 1 & inlist %group%1% %group%2% %group%3% %group%4%	1. skup5	pravda – jsou splněny všechny podmínky (počet hodnot je jedna, hodnota je jedna z hodnot atributu group)
ipAddr	valuescount 2 2 & contains ipv4 & contains ipv6	1. 192.168.80.5 2. fe80::6e88:14ff:fe55:3480	pravda – jsou splněny všechny podmínky (obsahuje právě dvě IP adresy, z nichž jedna je IPv4, druhá IPv6)

3.3.2 Filtrační řetězce

Všechny transformace dat jsou konfigurovatelné pomocí filtrů, respektive filtračních řetězců. Filtrační řetězec obsahuje filtrační pravidla. Vykonání každého pravidla je

podmíněno splněním validačních pravidel, která jsou jeho součástí. Formát filtračního řetězce znázorňuje syntaxe 3.3.

```
validační_pravidla ? filtrační_pravidla[ ; validační_pravidla ? filtrační_pravidla...]
```

Syntaxe 3.3: Syntaxe filtračního řetězce

Filtrační pravidla jsou oddělena čárkou. Budou vykonávána ve stejném pořadí, v jakém jsou definována. Stejně jako v případě validačních řetězců lze i filtrační podřetězce uzavírat do závorek. Tabulka 3.8 obsahuje seznam filtračních pravidel, která aplikace v základu obsahuje. Snadno lze vytvořit (naprogramovat) pravidla další.

Tab. 3.8: Filtrační pravidla

Pravidlo	Použití	Popis
trim	trim	Ořez prázdných znaků ze začátků a konců řetězců, které obsahují jednotlivé hodnoty atributu.
join	join left|right HODNOTA1 HODNOTA2 …	Ke všem hodnotám atributu přidat zleva (left) nebo zprava (right) hodnoty HODNOTA1, HODNOTA2, …
sequence	sequence NAZEV MIN MAX PRIRUSTEK	Nahradí hodnoty atributu sekvencí čísel. Sekvence je v databázi uložena pod názvem NAZEV a uchovává si číslo, které vrátí při dalším použití filtru. Toto číslo je v rozsahu od MIN do MAX a po každém použití filtru se inkrementuje o hodnotu PRIRUSTEK. Po překročení MAX se začíná znovu od MIN, případně pokud je číslo menší než MIN (PRIRUSTEK je záporná hodnota), pak se začíná znovu od MAX. Je to globální hodnota. Tu samou sekvenci lze použít u více atributů.
sendemail	sendemail ADRESA PREDMET TELO	Neprovádí žádnou transformaci hodnot atributů. Odešle e-mail na adresu ADRESA. Předmět e-mailu je PREDMET, tělo TELO.
setvar	setvar PROM HODNOTA	Vytvoří (nebo aktualizuje) v databázi proměnnou PROM a naplní jí hodnotou HODNOTA. Hodnotu proměnné lze použít ve validačním pravidle getvar.

Filtrační pravidla jsou svázána s konkrétními atributy či hodnotami. Argumenty pravidel nemusejí být statické hodnoty, pomocí proměnných je mohou z části nebo úplně tvořit hodnoty jiných atributů. Použití proměnných je stejné, jako u validačních pravidel.

Nelze spoléhat na to, že filtr bude mít k dispozici vždy hodnoty atributů identity. Filtry v závislosti na použití pracují s různými sadami atributů. Během aktualizace atributů identit pomocí atributů uživatelů koncového systému pracují filtry se sadou, ve které některé hodnoty atributů identit jsou nahrazeny atributy uživatelů koncových systémů. S tím je třeba počítat. Někdy se může stát, že bude v různých časech dvakrát filtrována stejná hodnota. Většinou není vhodné, aby po dvou průchodech tímtéž filtrem byla již jednou profiltrovaná ta samá hodnota znovu transformována.

V případě transformace jména identity na nové jméno uživatele během jeho vytváření v koncovém systému nemá filtr k dispozici žádnou sadu atributů, pouze virtuální atribut obsahující název identity.

Příklad použití

Tabulka 3.9 ukazuje příklady použití filtrů na různé atributy identity. Ve sloupcích obsahujících hodnoty atributů jsou u hodnot čísla určující pořadí, ve kterém jsou hodnoty uloženy v databázi.

Tab. 3.9: Příklady použití filtračních pravidel

Název atributu	Filtrační řetězec	Popis filtračních pravidel	Hodnoty před filtrací	Hodnoty po filtraci
fullName	any ? trim	Všechny hodnoty atributu budou mít ořezány koncové mezery.	1. Uc Li	1.Uc Li
uidNumber	inlist -1 ? sequence uidnumber 10000 200000 1	Každá hodnota atributu, která je -1 (může to být jeho výchozí hodnota) se nahradí hodnotou, kterou obsahuje sekvence uidnumber. Minimální hodnota sekvence je 10000, maximální 200000. Inkrementuje se o 1.	1. -1	1. 15187
seat	any ? sequence s 1 5000 2	Všechny hodnoty atributy budou nahrazeny definovanou sekvencí (1 až 5000, inkrementace po 2).	1. 500 2. -91 3. 45	1. 1 2. 3 3. 5

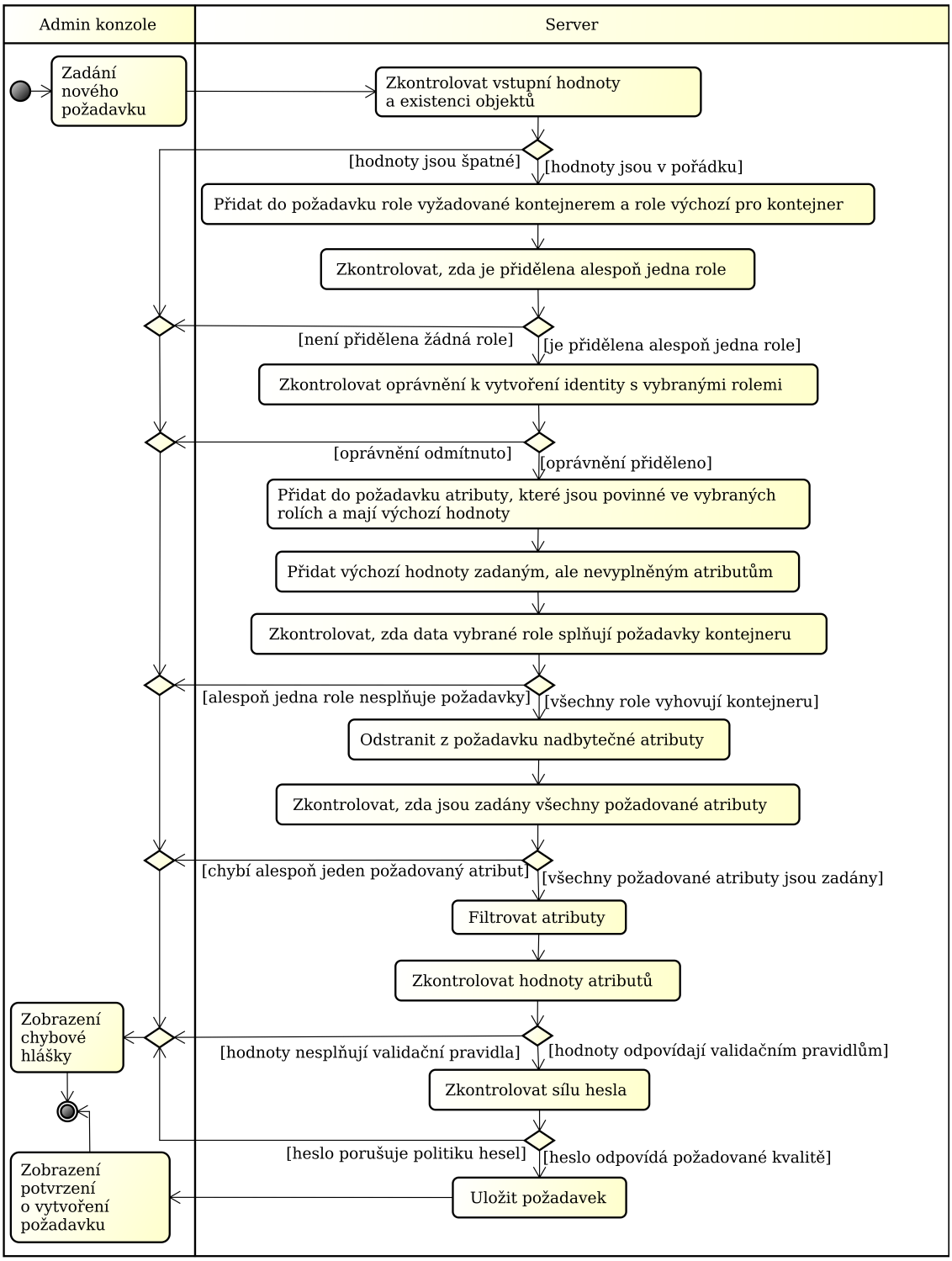
Název atributu	Filtrační řetězec	Popis filtračních pravidel	Hodnoty před filtrací	Hodnoty po filtraci
badPwdCount	valuescount 1 1 & range 3 2000000 & look mail valuescount 1 1 & not getvar badpwdcountsent inlist yes ? sendemail %mail%1% Upozornění Počet_přihlašovacích_pokusů_je_větší_než_2\nJestliže_jste_zapomněli_heslo\,_kontaktujte_admina, setvar badpwdcountsent yes; valuescount 1 1 & getvar badpwdcountsent inlist yes & range 0 2 ? setvar badpwdcountsent no	Filtrační pravidla neprovádějí žádnou transformaci. Jestliže je hodnota atributu badPwdCount větší nebo rovna hodnotě 3 a identita má nastavenou e-mailovou adresu a pouze tehdy, když proměnná badpwdcountsent neobsahuje yes, pak systém zašle uživateli e-mail. Po zaslání e-mailu se nastaví proměnná badpwdcountsent na hodnotu yes. Díky tomu se nebude po každé filtrací hodnot odesílat nový e-mail. Pokud hodnota atributu klesne pod 3, pak se proměnná badpwdcountsent nastaví na řetězec no. Díky setvar a getvar si tedy aplikace může pamatovat různé stavy.	1. 4	1. 4
mail	valuescount 1 1 & not email ? join right @f.cz	Jestliže hodnota atributu mail není platná e-mailová adresa, pak k ní zprava přidám řetězec @f.cz. Pro praktické použití by filtrační řetězec potřeboval samozřejmě vylepšit, protože přidáním doménové části nemusí vzniknou platný e-mail.	1. a	1. a@f.cz

3.4 Postupy vytváření požadavků

Po přihlášení do aplikace vytvoří administrátor požadavek. Součástí každého požadavku je popis a důvod. Popis by měl jednoznačně definovat vytvářený požadavek tak, aby ho schvalovatelé snadno identifikovali. V důvodu by mělo být uvedeno, proč byl požadavek vytvořen. Další součásti požadavku jsou závislé na konkrétnímu typu vytvářeného požadavku. Během ukládání každého požadavku se kromě kontroly hodnot a oprávnění zjišťuje, zda může být reálně proveden.

3.4.1 Vytvoření požadavku na přidání identity

Obrázek 3.2 znázorňuje kroky, které aplikace vykoná před tím, než je požadavek uložen a může být zpracováván. Zobrazuje akce provedené jak na straně klienta (rozhraní příkazového řádku pro administrátory), tak na straně serveru.



Obr. 3.2: Postup vytvoření požadavku na vytvoření identity

Požadavek obsahuje kromě popisu a důvodu následující: název vytvářené identity, heslo, kontejner, ve kterém bude identita umístěna, přiřazené role a atributy s hodnotami.

Po odeslání dat se zkontroluje formát vstupních hodnot a prověří se, zda existují vybrané role, schémata zadaných atributů a požadovaný kontejner. Poté se zjistí, zda název zvolené identity v systému již neexistuje. Na základě nastavení kontejneru se ověří, zda byly vybrány všechny kontejnerem vyžadované nebo pro nové objekty výchozí role. V případě potřeby se do požadavku chybějící role doplní. Nová identita musí mít přidělenou alespoň jednou roli. V tomto bodě proběhne kontrola oprávnění. Bude se zjišťovat, zda uživatel vytvářející požadavek má oprávnění vytvořit identitu se všemi přiřazenými rolemi. Během této operace se nahrají schvalovací procesy.

Nyní má aplikace k dispozici dostatek informací, aby na základě rolí vyhodnotila, zda byly zadány všechny povinné atributy. Pro každý chybějící povinný atribut zjistí, zda jsou v jeho schématu definovány výchozí hodnoty. Pokud jsou, pak je daný atribut přidán do požadavku s jeho výchozími hodnotami. V požadavku se mohou vyskytovat atributy, kterým nebyly uživatelem (administrátorem) přiřazeny žádné hodnoty. Všem těmto atributům se aplikace pokusí přiřadit výchozí hodnoty (pokud byly definovány) dle jejich schémat.

Poté aplikace zkontroluje, zda byly splněny požadavky kontejneru. Tímto krokem zabrání tomu, aby identita obsahovala role, které nejsou v kontejneru povolené.

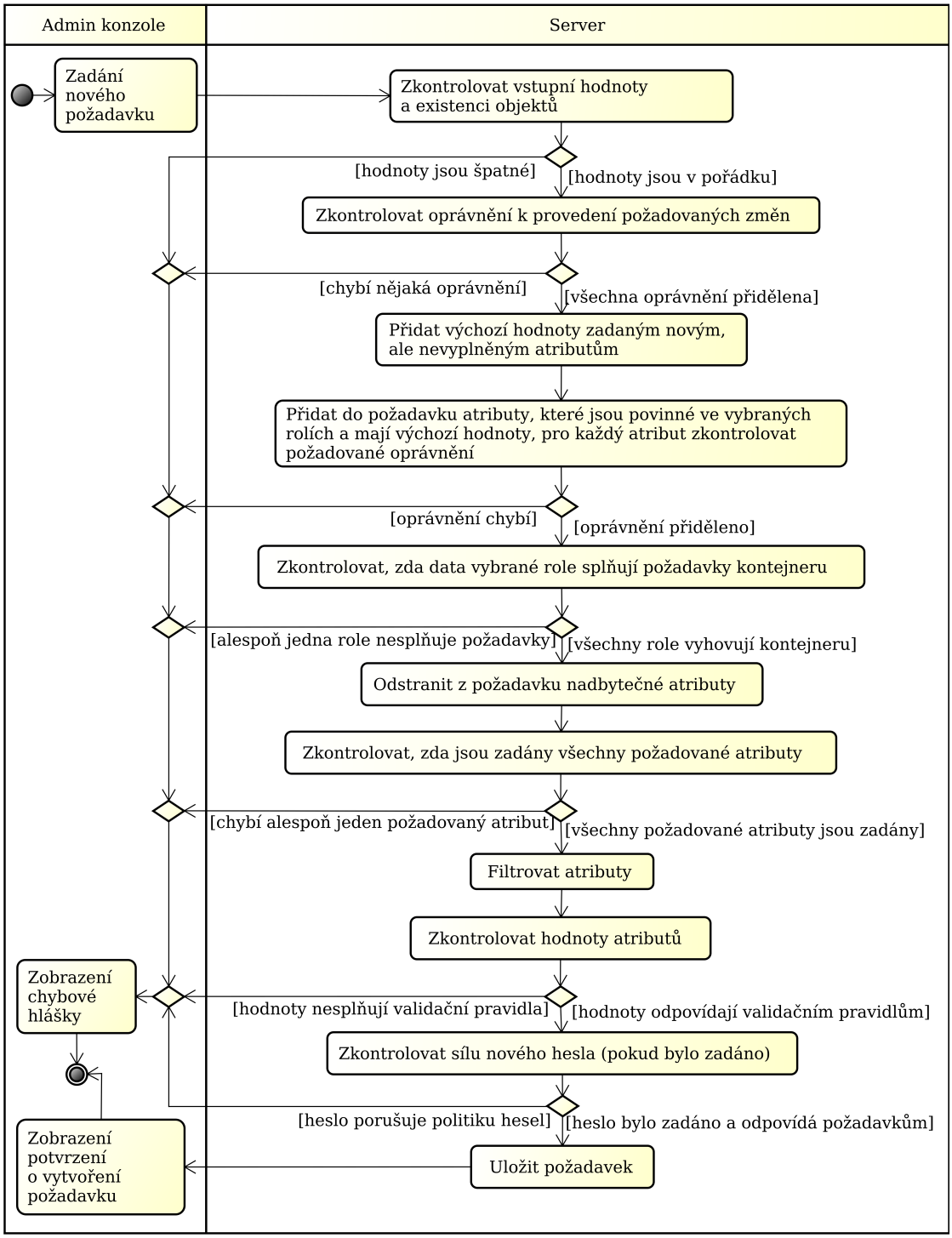
V dalším kroku odstraní aplikace z požadavku všechny nadbytečné atributy, tzn. takové atributy, které nejsou definovány v žádné přidělované roli. Pokud nechybí žádné povinné atributy, pak se provede transformace hodnot na základě filtračních pravidel definovaných pro jednotlivé atributy v jejich schématech. Poté se na základě validačních pravidel zkontrolují hodnoty atributů.

Na závěr aplikace zkontroluje sílu hesla zvoleného pro vytvářenou identitu a uloží požadavek do databáze k dalšímu zpracování. Uživatel, který požadavek vytvořil se dozví výsledek operace, další kroky probíhají v pozadí.

Spustí se schvalovací proces, po jehož dokončení se aplikace pokusí požadavek autorizovat, tj. vytvořit požadovanou identitu. Počítá se s tím, že proces schvalování může trvat delší dobu. Během této doby by mohlo dojít k úpravě schémat atributů, nastavení rolí apod. Proto se všechny kroky (kromě kontroly oprávnění) popsané v předchozích odstavcích provedou znovu během autorizace požadavku.

3.4.2 Vytvoření požadavku na úpravu identity

Obrázek 3.3 zobrazuje postup vytvoření požadavku na úpravu identity. Vzhledem k množství kroků je postup na obrázku zjednodušen.



Obr. 3.3: Postup vytvoření požadavku na úpravu identity

Požadavek kromě popisu a důvodu obsahuje povinně název upravované identity, volitelně:

- nový název identity (v případě přejmenovávání), který bude po filtraci použit i pro nové názvy všech koncových účtů,
- heslo, které bude nastaveno identitě a všem připojeným účtům v koncových systémech,
- změnu stavu z povoleno na blokováno nebo naopak,
- nastavení úrovně přístupu (uživatelské rozhraní nebo uživatelské a správcovské rozhraní),
- název kontejneru, do kterého má být identita přesunuta,
- seznam rolí, které mají být identitě přiděleny,
- seznam rolí, které mají být identitě odebrány,
- nové atributy, které budou součástí identity nebo jejichž hodnoty budou sloučeny s již existujícími atributy,
- hodnoty, které budou z existujících atributů odstraněny nebo názvy atributů, které budou identitě odebrány,
- atributy, jejichž hodnoty nahradí obsah existujících atributů.

Po odeslání dat se zkontrolují vstupní hodnoty a ověří se existence vybraných objektů, tzn. upravovaných identit, rolí pro odebrání nebo přidání, nového kontejneru a schémat všech uvedených atributů.

Poté podle vybraných požadovaných změn proběhne kontrola oprávnění a nahrávání schvalovacích procesů. Kontrola je mnohem komplexnější než u požadavku na vytvoření identity. Během ní se na základě členství cílové identity v různých rolích může ověřovat, zda uživatel vytvářející požadavek může měnit název identity, nastavovat heslo, přesunout identitu do jiného kontejneru, změnit status účtů (povolený/blokovaný), upravit nastavení přístupů k jednotlivým rozhraním aplikace, přidávat nebo odebírat členství ve vybraných rolích a upravovat hodnoty zvoleným atributům.

Po kontrole oprávnění se aplikace pokusí přidat výchozí hodnoty všem přidávaným nebo nahrazovaným atributům, jimž nebyly v požadavku nastaveny hodnoty za předpokladu, že identita zvolené atributy již neobsahuje.

Poté se prověří, zda byly zadány všechny atributy povinné v rolích, jejichž členem se po autorizaci požadavku identita stane. Všechny chybějící povinné atributy se aplikace pokusí do požadavku přidat. To se podaří pouze tehdy, mají-li atributy ve schématech uvedeny nějaké výchozí hodnoty a má-li uživatel vytvářející požadavek oprávnění k jejich úpravě.

Další postup je již identický s postupem vytváření požadavku na vytvoření identity. Ověří se splnění požadavků nového kontejneru, odstraní se nadbytečné atributy a provede se filtrace a kontrola hodnot všech atributů identity. Pokud je součástí požadavku nové heslo, provede se kontrola jeho síly.

Na závěr se požadavek uloží do databáze k dalšímu zpracování a jeho autorovi se zobrazí výsledek operace.

V pozadí se spustí schvalovací proces, po jehož dokončení se aplikace pokusí požadavek autorizovat, tj. upravit požadovanou identitu. Stejně jako v případě požadavku na vytváření identity se během autorizace provedou všechny kroky (kromě kontroly oprávnění) zmiňované v předchozím textu znovu.

3.4.3 Vytvoření požadavku na odstranění identity

Kromě popisu a důvodu obsahuje požadavek pouze název identity. Po zkontrolování vstupních hodnot a kontroly existence vybrané identity se na základě členství cílové identity v rolích zkontroluje oprávnění k provedení dané akce. Při kontrole oprávnění proběhne nahrání schvalovacích procesů. Po uložení se požadavek zpracovává stejným způsobem jako ostatní zmiňované požadavky.

3.4.4 Vytvoření požadavku na delegaci schvalování

Uživatel (identita) může všechny své schvalovací požadavky přesměrovávat na někoho jiného. Požadavek kromě popisu a důvodu obsahuje cílovou identitu a časový interval, po který bude přesměrování požadavků na zvolenou identitu v platnosti. Po odeslání požadavku se zkontrolují vstupní hodnoty a existence vybrané identity. Jestliže aplikace

pro autora požadavku zpracovává v pozadí 5 a více dalších požadavků, pak se nový požadavek nepodaří vytvořit. Poté aplikace zkontroluje požadované oprávnění k vytvoření delegace a nahraje schvalovací procesy. Požadavek se uloží a zpracovává stejným způsobem jako ostatní zmiňované požadavky.

3.4.5 Vytvoření požadavku na zrušení delegace schvalování

Někdy se může stát, že je potřeba zrušit aktuálně delegované schvalování ještě před vypršením jeho platnosti. V takovém případě uživatel (identita) vytvoří požadavek na zrušení delegace schvalování. V jeden okamžik může být schvalování delegováno pouze na jednu identitu, proto je v požadavku uveden pouze popis a důvod. Požadavek lze vytvořit pouze tehdy, pokud již aplikace v pozadí nezpracovává 5 a více požadavků od daného autora. Po zkontrolování oprávnění k odstranění delegovaného schvalování a nahrání schvalovacích procesů se požadavek uloží do databáze. Poté se zpracovává stejným způsobem jako ostatní požadavky.

3.5 Zpracování požadavků

Požadavky mohou nabývat dvou stavů:

- STATE_PENDING_CHANGES, kdy se čeká na zpracovatele požadavků,
- STATE_WAITING_FOR_APPROVAL, kdy se čeká na schválení kroku schvalovacího procesu.

Nové požadavky se ukládají se stavem STATE_PENDING_CHANGES. Po nějaké době je zaregistruje zpracovatel požadavků pravidelně se spouštějící v pozadí. Načte jejich schvalovací proces a spustí časovač (pokud není spuštěn) určující jejich životnost. Kromě nových požadavků existují již probíhající požadavky se stavem STATE_PENDING_CHANGES. Pokud jsou požadavky zamítnuté, odstraní je z databáze. V případě, že schvalovací proces obsahuje nějaké kroky, pak načte nový krok, ve kterém v případě potřeby nahradí některé schvalovatele delegovanými identitami. Pokud žádné kroky neobsahuje, pak zpracovatel požadavek bere jako schválený a provede autorizaci, respektive vykonání požadavku. V opačném případě uloží požadavek do databáze se stavem STATE_WAITING_FOR_APPROVAL a čeká na schválení schvalovateli aktuálního kroku.

Poté kontroluje expirované požadavky ve stavu STATE_WAITING_FOR_APPROVAL a podle konfigurace schvalovacího procesu provádí požadované operace (schválení, zamítnutí).

Jakmile schvalovatel provede schválení nebo zamítnutí požadavku, uloží se ve stavu STATE_PENDING_CHANGES a zpracovatel s ním nakládá stejně jako s novým požadavkem.

3.6 Postup komunikace konektoru s aplikací

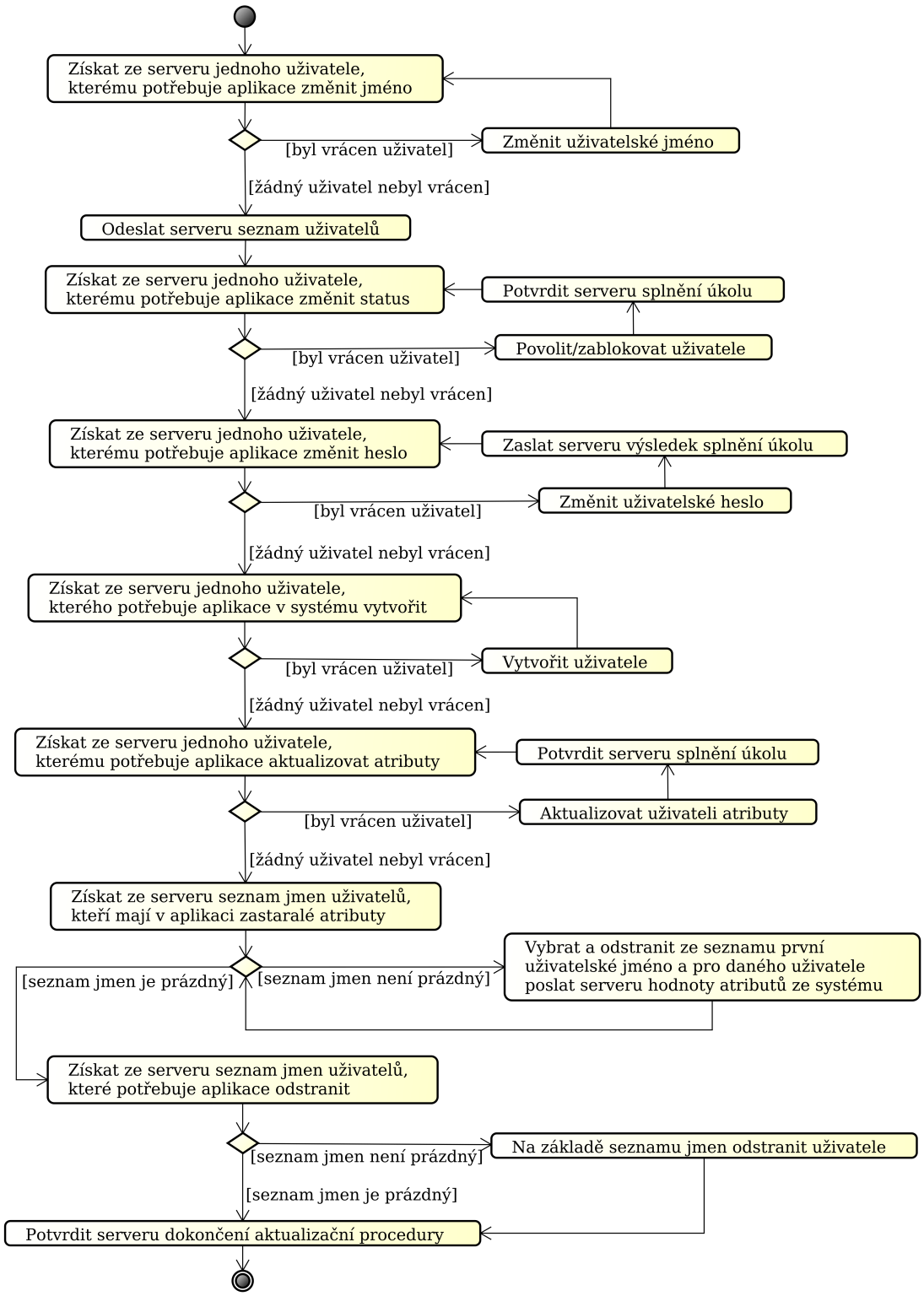
Centrální serverová aplikace nikdy nenavazuje spojení s žádným koncovým systémem. Navazování spojení je v režii konektorů. Konektor se vždy připojuje k aplikaci, nikdy aplikace ke konektoru. Díky tomu je práce s aplikací rychlá, při správě identit se nečeká na koncové systémy a nevadí ani jejich dlouhodobá nedostupnost.

Konektor pomocí JSON-RPC volá metody webového rozhraní serverové aplikace. Jako první metodu pro stažení přihlašovacího tokenu. Pouze zde se autentizuje klíčem uloženým v aplikaci pro konkrétní systém, který konektor spravuje. Při volání dalších metod používá k autentizaci stažený token. Pro bezproblémovou funkci musí dodržovat pořadí volání metod určených ke správě účtů.

Se serverem neudržuje trvalé spojení. V každém spojení zavolá vzdálenou metodu a zpracuje navrácená data. Konektor neběží trvale, interval a způsob jeho spouštění určuje správce systému. Po spuštění začne postupně volat různé serverové metody. Jestliže dodrží pořadí jejich použití, pak případný výpadek spojení během komunikace konektoru se serverem nezpůsobí trvalou nekonzistenci dat ani na straně serveru, ani na straně konektoru.

Serverová aplikace si v databázi udržuje záznamy všech uživatelských účtů, které se nacházejí v jednotlivých spravovaných systémech. Pro každý účet má lokálně uložené hodnoty všech atributů, které jí konektor spravující systém odešle. Jedná se vlastně o entity, kdy každá entita představuje jeden účet na koncovém systému.

Diagram na obrázku 3.4 znázorňuje princip činnosti konektoru. Ukazuje, jak probíhá komunikace mezi konektorem a serverem.



Obr. 3.4: Princip činnosti konektoru

1. Dokud bude aplikace vracet nějaký uživatelský účet, bude konektor v cyklu provádět následující operace. Požádá o zaslání jednoho uživatele, kterému potřebuje aplikace

změnit uživatelské jméno v koncovém systému. Aplikace nejprve změní název účtu ve své databázi v uložené uživatelské entitě. Poté vymaže žádost a odešle konektoru název uživatelského účtu spolu s novým názvem uživatele. Je to relativně nebezpečná operace. Výpadek spojení může způsobit, že si sice aplikace vede účet pod novým názvem, ale v koncovém systému nebyl účet přejmenován. Díky tomu si bude aplikace v následujících krocích myslet, že účet v koncovém systému byl vymazán a zároveň přibyl účet nový. Během následujících kroků však sebe sama uvede do konzistentního stavu.

2. Konektor odešle serveru seznam všech uživatelů, kteří se nacházejí na jím spravovaném systému. Seznam obsahuje u každého uživatelského účtu nějakou kontrolní hodnotu, která jednoznačně určuje aktualitu hodnot jeho atributů v koncovém systému. Touto hodnotou může být cokoliv. Typicky se bude jednat o nějaký časový údaj určující, kdy byl uživatel aktualizován. Hodnotou může být i hash získaný z hodnot atributů nebo nějaký náhodný text. Na základě této hodnoty aplikace určuje, kdy je potřeba aktualizovat systémem mapované atributy hodnotami z koncového systému.

   Aplikace (na serveru) projde všechna zaslaná uživatelská jména a vytvoří pro ně ve své databázi místní entity. Pokud již entita existuje, aktualizuje její kontrolní hodnotu. Poté smaže všechny místní entity (ne identity) představující koncové účty, které již ve spravovaném systému neexistují – nebyly v zaslaném seznamu.

3. Konektor požádá o zaslání jednoho uživatele, který má být v koncovém systému zablokován nebo povolen. Všechny místní entity představující koncové uživatele obsahují informaci, zda je koncový účet zablokován nebo povolen. Stav identity je distribuován všem namapovaným koncovým účtům. Serverová aplikace vrátí uživatelské jméno a požadovanou akci pro takový účet, jehož stav se liší od toho, ve kterém je namapovaná identita. Po provedení změn potvrdí konektor serveru vykonání akce. Jestliže operace na straně konektoru selhala a neodešle serveru potvrzení, pak konektor nemůže pokračovat v další činnosti. Všechny předchozí kroky se budou opakovat do té doby, dokud bude serverová aplikace vracet nějaký uživatelský účet.

Způsob reakce konektoru na příkaz „zablokuj účet“ nebo „povol účet“ závisí pouze na konektoru a aplikace počítá s tím, že u některých systémů nemusí jít aktuální stav snadno zjistit. Proto neimplementuje žádný způsob na zjišťování aktuálního stavu účtů v koncových systémech. Pokud bude účet v koncovém systému blokován nebo povolen jiným softwarem, pak se to aplikace nedozví a bude obsahovat nekonzistentní informace. Tento potenciální problém řeší tak, že v pravidelných intervalech znovu vrací stejné uživatelské účty s příslušným příkazem (blokovat/povolit).

4. Dokud bude server vracet nějaký uživatelský účet, bude konektor opakovat následující sekvenci. Požádá serverovou aplikaci o zaslání jednoho uživatele, kterému má změnit heslo ve spravovaném systému. Aplikace vrátí konektoru uživatelské jméno a nové heslo. Konektor se pokusí nastavit heslo uživateli spravovaného systému. Poté pošle serveru informaci o výsledku provedené akce (úspěch/neúspěch). Server si výsledek akce uloží a nezávisle na úspěchu vymaže žádost o nastavení nového hesla. Díky tomu se o případném neúspěchu může dozvědět příslušná identita nebo administrátor a není nutné dlouhodobě skladovat zvolená hesla v databázi.
5. Dokud bude serverová aplikace vracet nějaké účty, bude konektor opakovat následující postup. Požádá server o zaslání jednoho uživatelského účtu, který je potřeba v koncovém systému vytvořit. Pokud nemá v aplikaci nakonfigurovaný systém nastavené ignorování neexistujících uživatelů, pak serverová aplikace vrátí název uživatelského účtu, heslo (pokud ho koncový systém podporuje) a mapované atributy. Aplikace vybere takovou identitu, která má mít v koncovém systému účet a žádný takový účet není na identitu namapován. Identita musí splňovat podmínku, že čas její aktualizace je nižší než čas aktualizace daného systému. Tím se zajistí, že proběhlo mapování účtů daného systému, protože čas úpravy identity je starší než čas aktualizace dat z konektoru.

   V některých případech může po transformaci uživatelského jména identity aplikace zjistit, že požadovaný účet v koncovém systému již existuje. Pokud takový účet není ještě namapován na žádnou identitu, pak aplikace provede mapování a upozorní konektor. Pokud již namapován je, pak nastala kolize jmen. Tento stav musí vyřešit

správce systému. Příčinou může být nevhodné nastavení pravidel transformujících název identity do názvů účtů v koncových systémech.

Pokud koncový účet ještě neexistuje, pak pro něj ve své databázi serverová aplikace vytvoří prázdnou entitu. Taková entita obsahuje pouze název účtu, informaci o propojení (mapování) na identitu a vypnutý příznak určující, že jde o první synchronizaci dat z koncového systému do aplikace. Díky tomu se v dalších krocích nebude aplikace pokoušet aplikovat pravidla první synchronizace dat. Po vytvoření entity odešle aplikace konektoru požadovaný účet. Ten bude obsahovat takové atributy, které jsou do systému namapované. Nemusejí mít zapnutou synchronizaci směrem do systému. Hodnoty všech těchto atributů projdou transformačními pravidly. Aplikace nevyžaduje po konektoru potvrzení vytvoření koncového účtu. Předpokládá úspěch. V případě neúspěchu může na krátkou dobu obsahovat nekonzistentní informace. Tento stav automaticky vyřeší další spuštění konektoru.

6. Konektor požádá server o zaslání jednoho účtu, kterému je zapotřebí aktualizovat atributy v koncovém systému. Každý atribut identity obsahuje číselnou informaci určující verzi dat. Informace neurčuje verzi hodnot konkrétního atributu, po změně hodnot atributu bude jeho verze nastavena na maximální verzi všech atributů identity zvýšenou o jedničku. Entity koncových účtů obsahují číselný údaj určující verzi atributů identity, se kterou byl účet směrem do systému synchronizován. Všímá si pouze těch atributů, které jsou mapované a synchronizované směrem do systému. Konektor vrátí takový účet, jehož místní entita má tento údaj nižší než je maximální hodnota všech do systému namapovaných atributů, které mají povolenou synchronizaci směrem do systému. Před zasláním účtu konektoru nastaví aplikace tento údaj tak, že bude obsahovat maximální hodnotu všech do systému namapovaných atributů vynásobenou hodnotou -1.

   Informace zaslané konektoru obsahují název uživatelského účtu a hodnoty atributů jeho identity namapovaných s nastavenou synchronizací do koncového systému. Všechny odeslané hodnoty prošly transformačními pravidly.

   Po provedení změn odešle konektor serveru potvrzení. Serverová aplikace nahradí v příslušné entitě hodnotu obsahující zápornou maximální verzi synchronizovaného atributu její absolutní hodnotou. Tím se potvrdí synchronizace dat vůči jejich určité

verzi na straně serveru. Bez potvrzení provedené operace nemůže pokračovat konektor v další činnosti.

Celý postup popsaný v tomto bodě se bude opakovat do té doby, dokud bude server vracet nějaký uživatelský účet.

7. Konektor stáhne ze serveru seznam uživatelských účtů, u kterých má neaktuální hodnoty atributů. Server potřebuje, aby každému zaslanému účtu konektor aktualizoval hodnoty atributů v aplikaci hodnotami atributů ve spravovaném systému.

   Serverová aplikace vrátí v seznamu takové uživatelské účty, jejichž místní entita obsahuje novou kontrolní hodnotu určující aktualitu koncových atributů. Seznam bude obsahovat i účty, jejichž kontrolní hodnota byla aktualizovaná v předminulé aktualizaci prováděné ve 2. bodě. To ve výsledku způsobí, že každý zaslaný uživatelský účet bude aktualizován alespoň dvakrát (podruhé v dalším spuštění konektoru). Tím se zajistí konzistence hodnot atributů v aplikaci s hodnotami v konektoru i v případech, kdy kontrolní hodnota obsahuje nevhodné údaje jako čas poslední aktualizace koncového účtu. Mohlo by se stát, že konektor odešle hodnoty serverové aplikaci ve chvíli, kdy se v koncovém systému zrovna mění. Pokud se změna odehraje v jeden časový okamžik, pak by aplikace nezaregistrovala fakt, že hodnoty v koncovém systému byly znovu změněny.

   Konektor projde seznam zaslaných uživatelů. Postupně po jednotlivém uživateli pošle serveru hodnoty jeho atributů v koncovém systému.

   Jestliže je účet v aplikaci mapován na nějakou identitu a aplikace má k dispozici nové hodnoty atributů pro koncový systém (podle verze dat), pak je aktualizace hodnot atributů v aplikace odmítnuta. K této situaci může dojít, pokud jsou atributy identity aktualizovány po provedení 6. bodu a před vykonáním operací v tomto bodě. Synchronizace hodnot z aplikace do koncového systému má vždy přednost před synchronizací hodnot z koncového systému do aplikace.

   Pokud konektor nezašle všechny mapované atributy, aplikace zachová jejich původní hodnoty. Pokud konektor zašle některé atributy bez hodnot, aplikace je místní entitě odebere. Jak již bylo zmiňováno entita představuje jakýsi zrcadlený účet koncového systému. Po aktualizaci hodnot místní entity se zkontroluje, zda je účet (entita)

namapován na nějakou identitu. Pokud není, pak aplikace postupně projde všechny identity, které by měly být namapovány na účty systému, se kterým konektor právě pracuje. Pokud lze na základě nastavených parametrů provést spojení účtu s identitou, aplikace provede mapování, zapne příznak první synchronizace a zruší prohledávání dalších identit.

Jestliže je entita namapována na identitu, provede aplikace na základě konfigurace a po transformaci (filtraci) aktualizaci hodnot atributů identity pomocí atributů koncového účtu (entity). Pokud se jedná o první aktualizaci (synchronizaci) dat, řídí se aplikace nastavenými pravidly. Poté aplikace synchronizuje entitě hodnotu verze mapovaných a synchronizovaných atributů s jejich skutečnou poslední verzí. Bez tohoto kroku by docházelo k zacyklení – systém by aktualizoval identitu, identita účet systému, …

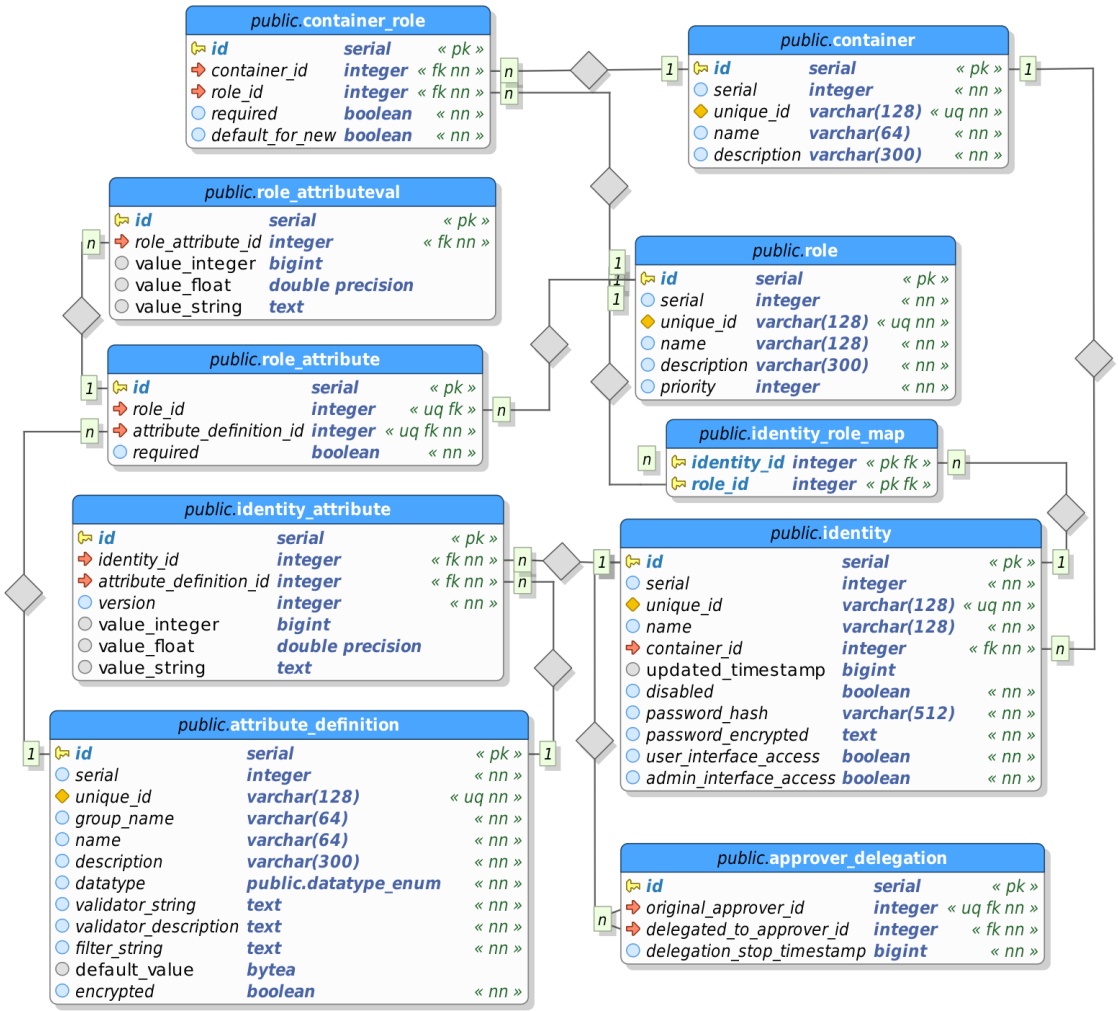
8. Konektor se připojí k serverové aplikaci a stáhne seznam jmen uživatelů, které je potřeba odstranit ve spravovaném systému. Po každém odstranění identity se na všech na identitu mapovaných účtech spustí časovače. Po uplynutí časovačů se účty stanou neplatnými. Aplikace vrátí všechny neplatné uživatelské účty. Dle naprogramování konektoru může být akce odstranění účtů provedena různě – od prostého zablokování až po skutečné odstranění. Není potřeba, aby konektor potvrzoval aplikaci provedení operace. Po dalším spuštění konektoru se aplikace výsledek dozví na základě zaslaného seznamu uživatelských jmen.
9. Konektor se připojí k serverové aplikaci a potvrdí provedení celé série výše zmíněných bodů aktualizace. Aplikace aktualizuje čas aktualizace konektoru.

3.7 Datový model

Aplikace používá pro ukládání svých dat relační databázi PostgreSQL. Je závislá na způsobu práce s touto databází, na databázových triggerech a na konkrétní implementaci transakcí. Pro všechny transakce používá aplikace úroveň izolace Repeatable Read. Její implementace v PostgreSQL je přísnější, než specifikuje standard. Jak uvádí dokumentace [14], zabraňuje výskytu fantomů. Díky této úrovni izolace musí návrh aplikace počítat s pravděpodobným výskytem selhání transakcí. Takové transakce musí aplikace opakovat.

Diagram databáze je kvůli lepší přehlednosti rozdělen do tří obrázků. Díky tomu nejsou zobrazeny všechny vazby. Zobrazuje přímo tabulky, názvy sloupců a datové typy používané databázovým strojem PostgreSQL. Tam, kde je to možné, jsou mezi tabulkami znázorněny vzájemné vazby včetně kardinality. Podle názvů cizích klíčů (většinou tabulka_sloupec) si lze chybějící vazby domyslet. Některé tabulky obsahují sloupce *serial* a *unique_id*. Využívají se při mapování objektů na relace (ORM). Podle hodnoty *serial* dokáže aplikace rozeznat, zda objekt od jeho načtení námi neuložil někdo jiný. Hodnoty sloupce *unique_id* jednoznačně identifikují objekt v rámci celé aplikace.

Na obrázku 3.5 jsou znázorněny tabulky, které mají úzký vztah s identitami a jejich organizací.



Obr. 3.5: Diagram databáze – identity a jejich organizace

Tabulka ***attribute_definition*** slouží pro ukládání schémat atributů. Jejich výchozí hodnoty jsou uložené v binární formě ve sloupci *default_value*. Identity ukládá aplikace do tabulky ***identity***. Heslo identit je uloženo ve dvou sloupcích. Ve sloupci *password_hash* v podobě hashe a ve sloupci *password_encrypted* v zašifrované formě. Aplikace ověřuje hesla identit na základě jejich hashe. Nová hesla ukládá se solí za pomocí hashovací funkce SHA512, rozumí ale i jiným formátům (včetně formátu prostého uložení otevřeného hesla), které dokáže detekovat. Zašifrovanou formu hesla používá při zakládání koncových účtů. Tento návrh umožňuje případný snadný přechod z jiných softwarů.

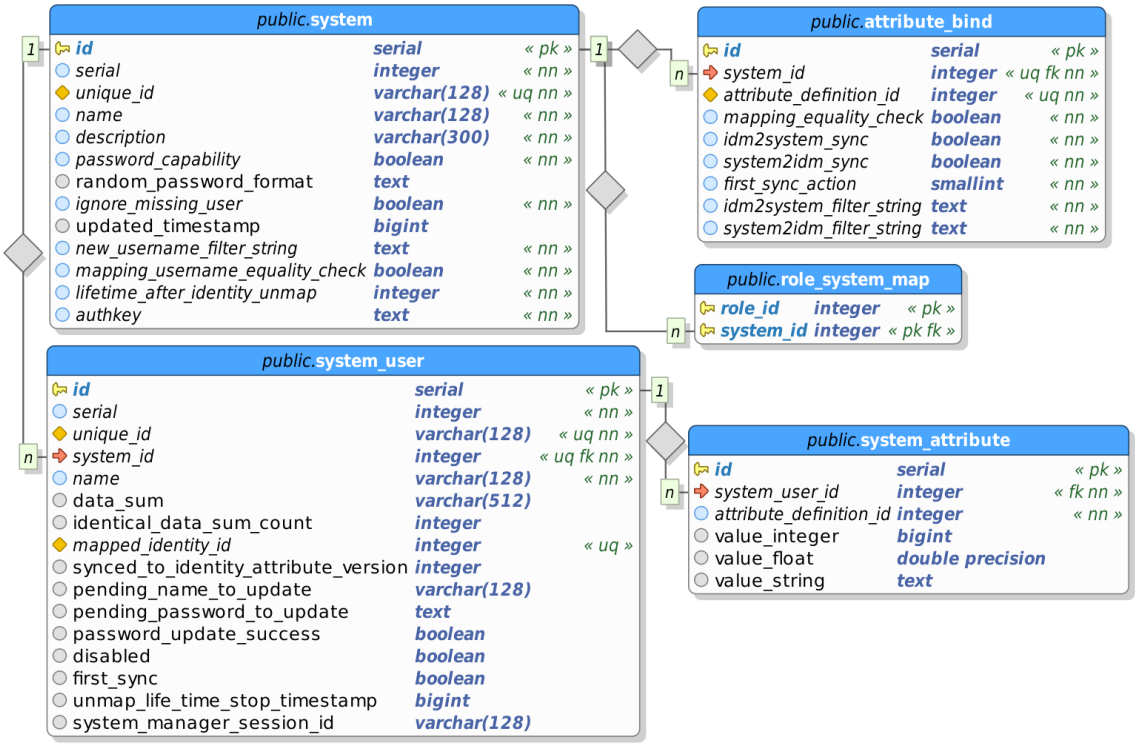
Hodnoty atributů identity se nacházejí v tabulce ***identity_attribute***. Na základě datového typu atributu se ukládají do sloupce *value_integer*, *value_float* nebo *value_string*. Datový typ INTEGER a BOOLEAN do *value_integer*, FLOAT do sloupce *value_float*, BINARY, CISTRING a STRING do *value_string*. BINARY je před uložením do textového sloupce převeden do formátu base64. Všechny hodnoty atributů s nastaveným šifrováním se nezávisle na datovém typu po zašifrování a zakódování pomocí base64 ukládají do *value_string*.

Tabulka ***approver_delegation*** obsahuje informace o delegování schvalovacích procesů. Ve sloupci *delegation_stop_timestamp* je datum a čas ve formě unix timestamp, kdy daná delegace přestane být aktivní.

Definice rolí jsou ukládány do tabulky ***role***. Seznam atributů rolí je umístěn v tabulce ***role_attribute***. Pokud má role předdefinovány nějaké hodnoty atributů, pak jsou uloženy v tabulce ***role_attributeval*** stejným způsobem jako v případě hodnot atributů identit. Přiřazení rolí k identitám řídí hodnoty v tabulce ***identity_role_map***.

Objekty kontejnerů jsou ukládány do tabulky ***container***. Seznam všech rolí, které může kontejner obsahovat včetně konfigurace jejich použití je umístěn v tabulce ***container_role***.

Na obrázku 3.6 je část diagramu znázorňující, jakým způsobem se ukládá konfigurace systémů a koncové uživatelské účty.



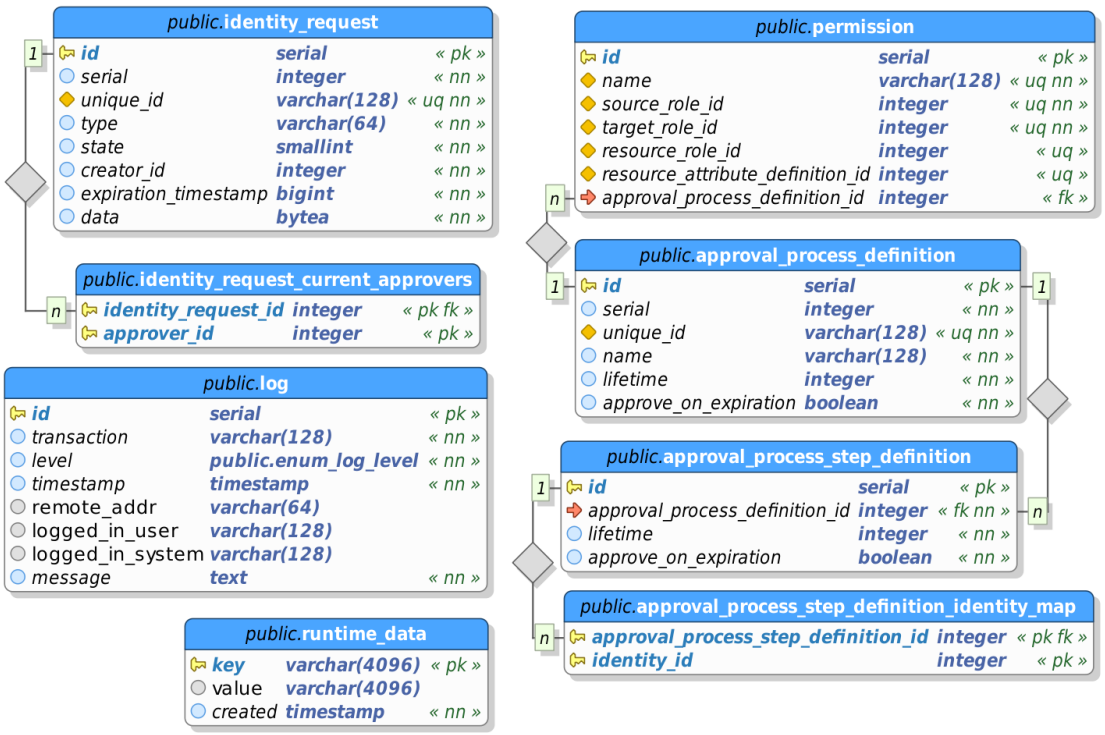
Obr. 3.6: Diagram databáze – nastavení systémů a ukládání koncových účtů

Účty načtené z koncových systémů (místní entity) se ukládají do tabulky ***system_user***. Sloupec *data_sum* obsahuje kontrolní hodnotu určující aktualitu atributů koncového účtu, *identical_data_sum_count* ukládá hodnotu čítače, pomocí kterého může aplikace zjistit, jak dlouho (v počtu zpracování seznamu zaslaných účtů) konektor zasílá stejnou kontrolní hodnotu. Sloupec *mapped_identity_id* ukládá *id* mapovaných identit. Sloupec není definovaný jako cizí klíč. Po odstranění identity nebo role potřebné pro udržení mapování nastavují hodnotu sloupce *mapped_identity_id* na NULL vytvořené triggery, které zároveň vykonají další potřebné operace (nastavení časovače odstranění účtu, ...). Podle hodnot ve sloupci *pending_name_to_update* a *pending_password_to_update* pozná konektor, kdy je zapotřebí změnit jméno či heslo koncovému spravovanému účtu. Na základě hodnoty ve sloupci *password_update_success* lze poznat případné selhání operace změny hesla koncového účtu. Hodnoty atributů koncového účtu jsou ukládány v tabulce ***system_attribute***. Ukládají se stejným způsobem jako hodnoty atributů identity.

Nastavení systémů se ukládá do tabulek ***system*** a ***attribute_bind***. Tabulka ***system*** obsahuje hlavní nastavení chování systému (respektive aplikace k systému a naopak), nastavení mapování atributů uchovává tabulka ***attribute_bind***. Konfiguruje se, jaké

atributy aplikace koncový systém (konektor) vidí a jak se s nimi zachází. Systémy, v nichž bude mít identita účet se určují na základě členství identity v rolích a mapování rolí na systémy. To se uvádí v tabulce ***role_system_map***.

Poslední část diagramu je zobrazena na obrázku 3.7. Obsahuje tabulky, které slouží pro ukládání schvalovacích procesů, běžících schvalování, oprávnění a běhových informací.



Obr. 3.7: Diagram databáze – ukládání schvalovacích procesů a běhových informací

Oprávnění je součástí objektu rolí. V databázi se ukládá do tabulky ***permission***. Kromě zdrojové role (role, jejíž je součástí), cílové role a případného zdroje může oprávnění obsahovat schvalovací proces (*approval_process_definition_id*). Vytvořené schvalovací procesy ukládá aplikace do tabulky ***approval_process_definition***. Jejich kroky do tabulky ***approval_process_step_definition***. Každý krok obsahuje seznam schvalovatelů. Tabulka ***approval_process_step_definition_identity_map*** mapuje identity schvalovatelů na daný schvalovací krok.

Všechny aktivní požadavky jsou aplikací ukládané do tabulky ***identity_request***. Každý typ požadavku je do jisté míry unikátní, obsahuje různé proměnné a při každém schvalování se běžící požadavek musí celý načíst z databáze. Proto se ukládá ve formě serializovaného objektu do sloupce *data*. Aby aplikace mohla běžící požadavky

prohledávat, obsahuje tabulka další pomocné sloupce. Aplikace potřebuje prohledávat běžící požadavky i podle schvalovatelů. Tabulka ***identity_request_current_approvers*** obsahuje schvalovatele aktuálně vykonávaného schvalovacího kroku daného běžícího požadavku.

Podle nastavení úrovně logování ukládá aplikace mnoho užitečných informací o jejím běhu do tabulky ***log***. Podle sloupce *transaction* lze zjistit, kterého konkrétního spojení (klient-server komunikace) se uložený záznam týká. Alternativou ukládání logů do databáze by bylo použití prostých textových souborů. Tento způsob by však díky potřebě zamykání souborů značně zpomaloval běh celé aplikace. Podle potřeby si lze kdykoliv uložené logy exportovat do textových souborů. Jestliže dojde k havarijní situaci, aplikace danou chybu či výjimku uloží přímo do textového souboru.

Tabulka ***runtime_data*** obsahuje některá data, která si aplikace ukládá během svého běhu. Konkrétně se jedná o vygenerované přihlašovací tokeny uživatelů nebo data koncových systémů (respektive konektorů) a proměnné filtrů (setvar, sequence).

3.8 Aplikační třídy

Jenom hlavní serverová část aplikace obsahuje přes 100 tříd. Proto zde budou popisovány pouze vybrané třídy. Hlavním třídám jsou vkládány závislosti zvenčí buď pomocí konstruktorů nebo pomocí setter metod. Tomuto návrhovému vzoru se říká Dependency Injection (DI). Některé objekty jsou na sobě navzájem závislé, o složení stromu závislostí se stará aplikační kontejner. Principem práce DI kontejnerů se podrobně zabývá [15].

Hlavní aplikace nepoužívá žádné frameworky. Mapování relací databáze na objekty aplikace (ORM) řeší Manager třídy (IdentityManager, RoleManager, …). Instance takové třídy se stará o vytváření nových objektů, ukládání do databáze (perzistenci) a o jejich načítání z databáze do objektů aplikace. Díky tomu jsou Manager třídy jistou implementací návrhového vzoru Data Mapper. Více o tomto vzoru se zmiňuje [16]. Vzhledem k závislostem mezi objekty musí pomocí jiných Managerů načítat objekty, na nichž vytvářený nebo načítaný objekt závisí. Při tom se snaží načítat pouze takové závislosti, které jsou v danou chvíli nezbytné. Ostatní si načítají samostatně načtené objekty ve chvíli, kdy s nimi potřebují pracovat. Při ukládání objektů do databáze se

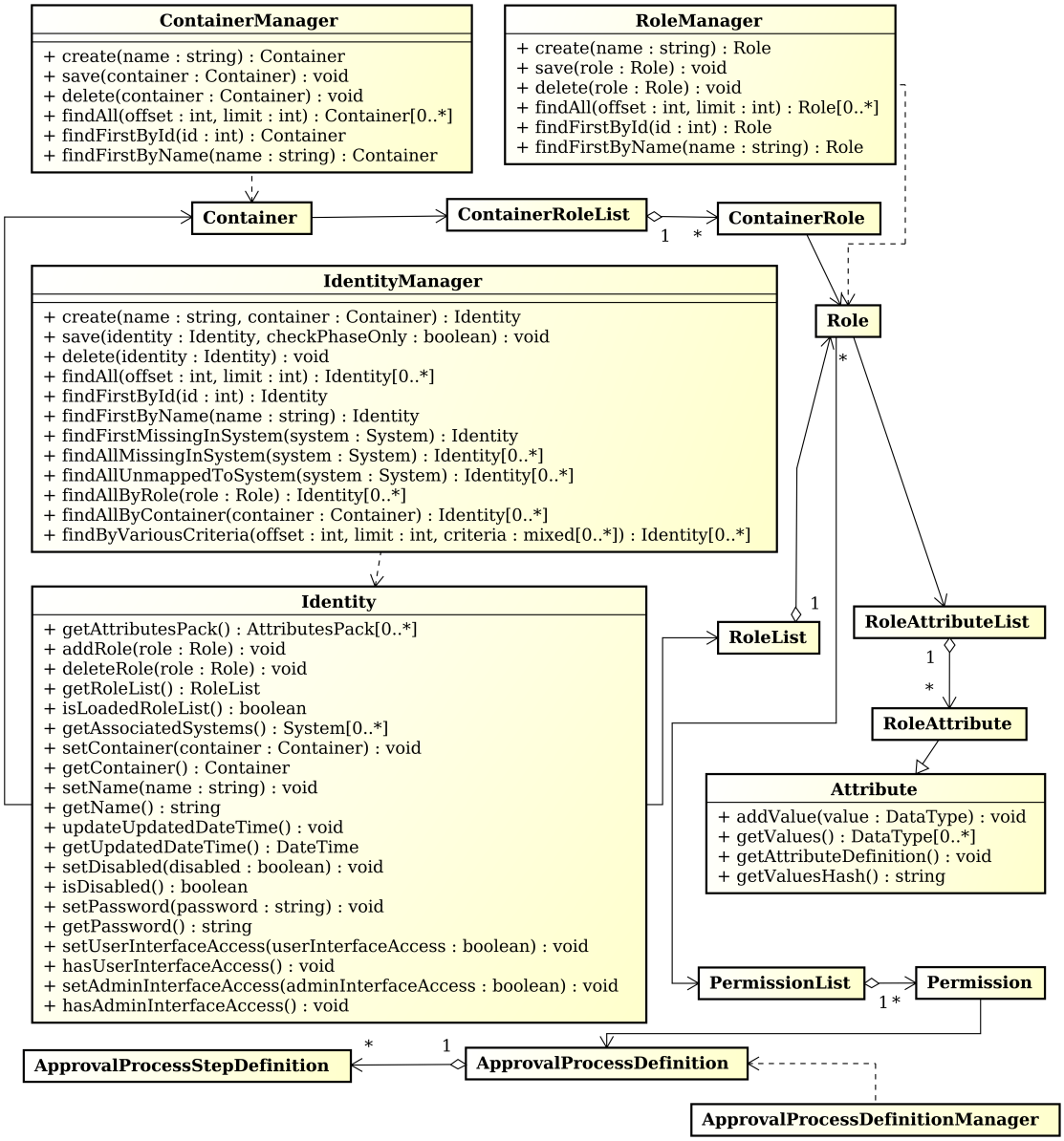
snaží aplikace ukládat pouze změněná data. Během ukládání se může pracovat s velkým množstvím objektů a databázových tabulek. Proto se provádí v transakci, případně několika vnořených transakcích. V případě selhání transakce nebo její vnořené části aplikace ví, kterou část operací musí znovu zopakovat proto, že již obsahuje neplatná data. Načtené objekty aplikace několik sekund udržuje ve vyrovnávací paměti. Jejich množství záleží na konkrétní Manager třídě a na tom, jak často se s daným objektem pracuje.

Kvůli úspoře místa nejsou v následujících diagramech uvedeny ani konstruktory, ani setter metody pro DI. Jsou zobrazeny pouze veřejné metody a některé závislosti mezi třídami. U některých tříd nejsou zobrazené žádné metody. Všechny třídy jsou součástí nějakých balíčků, které na těchto diagramech nejsou zobrazeny. Detailnější diagramy některých aplikačních tříd jsou k nahlédnutí na přiloženém DVD.

Aplikace vyžaduje implementaci nějakým interpretovaným programovacím jazykem bez silné typové kontroly. Je navržena objektově tak, aby jí šlo bez složitějšího programování upravovat, někdy i za běhu. Příkladem jsou transformační (filtrační) a validační pravidla.

Každé filtrační a validační pravidlo je představováno samostatnou třídou. Na základě názvů pravidel aplikace automaticky vyhledává a používá vhodné třídy. Podmínkou jsou správně umístěné soubory tříd v adresářové struktuře aplikace a vhodně nazvané soubory i třídy. Názvy musejí začínat velkým písmenem a pokračovat pouze písmeny malými.

Na obrázku 3.8 je znázorněn diagram některých důležitých tříd, které pracují s identitami, rolemi či kontejnery. Zvolené třídy nějakým způsobem mapují řádky několika tabulek databáze do výsledných objektů a naopak.



Obr. 3.8: Diagram tříd – identity, role, kontejner

Třída Identity

Třída ***Identity*** představuje identitu uživatele. Instance objektu obsahuje seznam rolí, kontejner, ve kterém je identita umístěna a objekt ***AttributesPack***, ve kterém jsou uloženy hodnoty atributů. Na základě rolí umožňuje zjistit asociované systémy, ve kterých by měla mít vytvořené účty. Seznam rolí je speciální kolekce – instance třídy ***RoleList***. Každá třída končící List je speciální druh kolekce. Taková kolekce je schopná si pamatovat, které objekty do ní byly přidány či odebrány. Dokonce umí porovnat obsah v ní umístěných objektů a zjistit, které objekty byly změněny. To přijde vhod během ukládání objektu do databáze. ***Identity*** načte kolekci ***RoleList*** pouze tehdy, když

je to potřeba. Pro načítání rolí používá ***RoleManager***. Metoda *getPassword* má speciální chování – nevrací aktuální heslo identity. Vrací heslo, které identitě nastavujeme metodou *setPassword*. Uložené heslo identity se načítá pouze na vybraných místech aplikace jiným způsobem a to pouze tehdy, je-li to potřeba.

Třída IdentityManager

Vytváření instancí třídy ***Identity*** má na starosti výhradně instance třídy ***IdentityManager***. Kromě toho umí ***IdentityManager*** z databáze odstraňovat data příslušející danému ***Identity*** objektu nebo data identit v databázi vyhledávat a vracet jako ***Identity*** objekty.

Třída Role

Instance třídy ***Role*** představuje roli v aplikaci. Obsahuje tři dynamicky načítané seznamy – instance tříd ***PermissionList***, ***RoleAttributeList*** a ***SystemList***. Oprávnění jsou součástí rolí, načítá je výhradně ***Role*** a ukládá ***RoleManager*** spolu s objektem ***Role***. Během načítání oprávnění načítá ***ApprovalProcessDefinitionManager*** odpovídající schvalovací procesy. ***RoleAttributeList*** je seznam objektů ***RoleAttribute***. Určuje, jaké atributy člen role získává, zda jsou povinné a definuje jejich fixní hodnoty. ***SystemList*** obsahuje seznam koncových systémů, které budou přiděleny členům role. Při načítání objektů ***System*** do ***SystemList*** využívá aplikace ***SystemManager***.

Třída RoleManager

Zajišťuje načítání rolí z databáze do objektů ***Role***. Stejně jako ostatní Manager třídy umí kromě načítání i jejich vyhledání, ukládání a odstraňování.

Třída Container

Instance ***Container*** třídy představuje kontejner, do kterého se ukládají identity. Obsahuje dynamicky načítaný seznam ***ContainerRoleList***, který obsahuje objekty ***ContainerRole*** určující, jaké role jsou v kontejneru povolené a jejich nastavení pro kontejner.

Třída ContainerManager

Stejně jako ***RoleManager***. Zajišťuje načítání a ukládání objektů ***Container***.

Třída Attribute

Instance třídy ***Attribute*** představuje atribut identity nebo účtu v koncovém systému. Obsahuje schéma atributu a jeho hodnoty. Identity obsahují většinou více než jeden atribut, proto jsou v objektu ***Identity*** jednotlivé instance ***Attribute*** součástí objektu ***AttributesPack***, se kterým se poté pracuje.

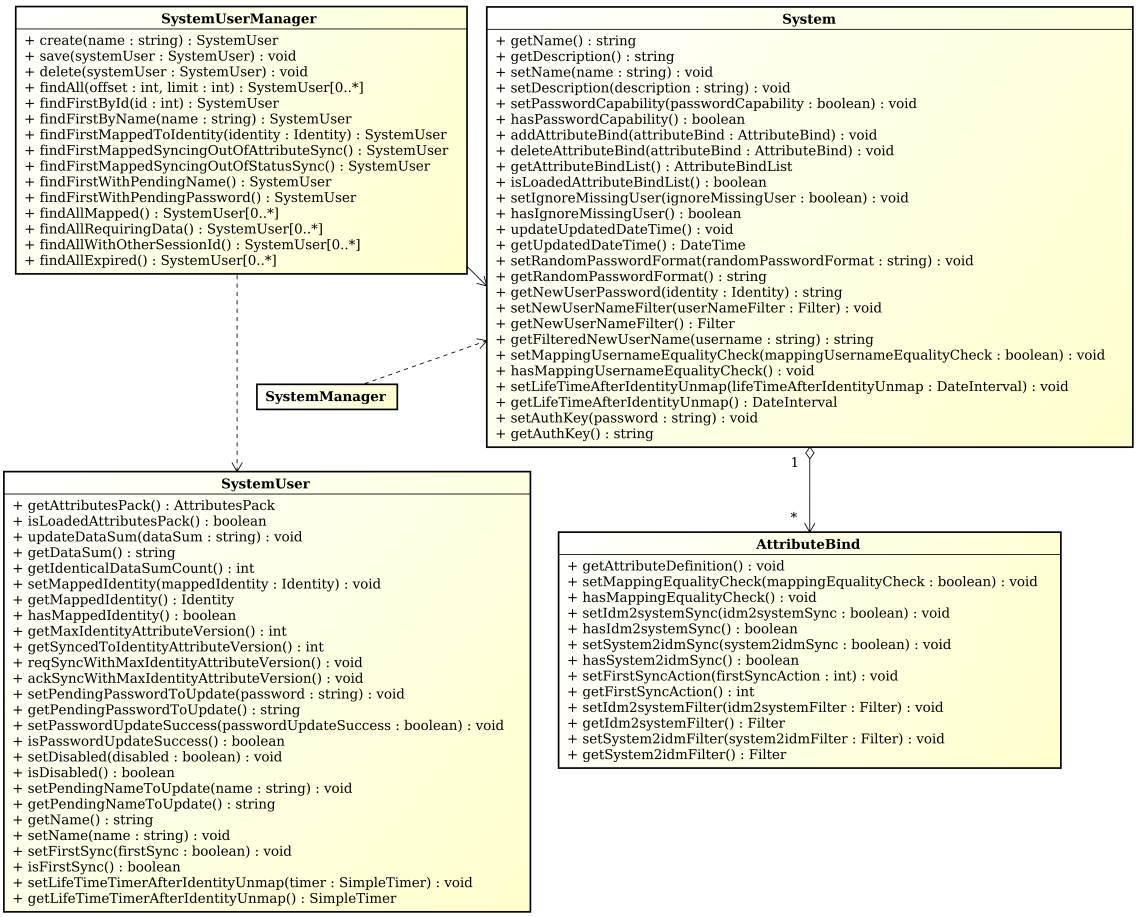
Třídy ApprovalProcessDefinition a ApprovalProcessStepDefinition

ApprovalProcessDefinition představuje konfiguraci schvalovacího procesu. Každý schvalovací proces je tvořen kroky. Každý krok je představován instancí třídy ***ApprovalProcessStepDefinition***.

Třída ApprovalProcessDefinitionManager

Plní stejnou funkci jako ostatní Manager třídy s tím, že jednotlivé schvalovací kroky (instance ***ApprovalProcessStepDefinition***) nejsou načítány dynamicky v seznamu. Počítá se s tím, že jsou nezbytnou součástí procesu, proto je načítá všechny najednou ***ApprovalProcessDefinitionManager*** při inicializaci ***ApprovalProcessDefinition***.

Diagram na obrázku 3.9 zobrazuje některé důležité třídy, které představují entity systémů a koncových uživatelů.



Obr. 3.9: Diagram tříd – systém a koncoví uživatelé

Třída System

Konfigurace propojení s koncovými systémy se ukládá do relační databáze a mapuje se na instance třídy ***System***. Na rozdíl od ostatních metod má metoda *getAuthKey* zvláštní chování. Vrací autentizační klíč systému, kterým se konektor přihlašuje k aplikaci pouze tehdy, byl-li právě nastaven metodou *setAuthKey*. Z databáze se uložený klíč nikdy nenačítá do objektů ***System***. Načtené objekty ***System*** obsahují dynamicky načítanou kolekci ***AttributeBindList***, jejíž prvky ***AttributeBind*** určují mapování atributů identit. ***AttributeBind*** určuje, jaké atributy identity budou přístupné koncovému systému a jakým způsobem s nimi může zacházet.

Třída SystemManager

Má stejnou funkci jako ostatní Manager třídy. Zajišťuje mapování relací do objektů ***System*** – vyhledávání, načítání a ukládání.

Třída SystemUser

Data všech uživatelských účtů v koncových systémech jsou ukládána do databáze aplikace a představována objekty ***SystemUser***. Instance třídy ***SystemUser*** obsahují atributy koncového účtu a představují vztah koncového účtu vzhledem k aplikaci. Ukládají si například následující hodnoty: nové heslo pro koncového uživatele, nové uživatelské jméno, výsledek nastavení hesla v koncovém systému, instance namapované identity aj.

Třída SystemUserManager

Instance ***SystemUserManager*** mapuje objekty ***SystemUser*** na relace databáze. Kromě běžných funkcí umožňuje vyhledávat ***SystemUser*** objekty podle různých kritérií: pouze objekty mapované na nějakou identitu, pouze objekty mající nové heslo pro koncového uživatele, …

4 Implementace aplikace

4.1 Použité technologie

Všechny části aplikace jsou psané v jazyce PHP. Tento jazyk splňuje požadavky kladené v návrhu – je interpretovaný a bez silné typové kontroly. Nevýhodou je, že stále prochází aktivním vývojem a jeho nové verze občas mění chování, na kterém je závislý některý dříve napsaný software. Při psaní aplikace jazykem PHP je vhodné vycházet z jeho aktuální oficiální dokumentace [17] a vyhýbat se použití některých experimentálních nebo zastaralých funkcí.

Data aplikace jsou ukládána v relační databázi PostgreSQL. Aplikace závisí na způsobu práce daného databázového stroje, na jeho vlastnostech a možnostech. Během časté práce s databází pracuje aplikace s velkým množstvím dat, je proto vhodné umístit hlavní aplikaci a databázový stroj na jeden server.

Hlavní serverová část aplikace vyžaduje ke své činnosti webový server. Je navržena a vyzkoušena tak, aby pracovala pod webovým serverem Apache httpd. Klienti (administrátoři, koncoví uživatelé, konektory) se připojují přes různá JSON-RPC rozhraní zveřejněná webovým serverem.

Administrátorské CLI (Command-Line Interface) rozhraní pro správce i vytvořený vzorový konektor používá pro zpracování skriptů CLI interpret jazyka PHP. Webové rozhraní pro koncové uživatele je klasická webová aplikace běžící pod webovým serverem.

4.2 Softwarové a hardwarové požadavky

Hlavní serverová část aplikace potřebuje ke své činnosti PHP verze 5.4.0 nebo vyšší a nějakou aktuálně podporovanou verzi PostgreSQL. Hardwarové požadavky serverové aplikace závisejí na mnoha faktorech. Především na množství spravovaných identit, na počtu konektorů a frekvenci jejich připojování, na množství a typu použitých atributů a na zvolených validačních a filtračních pravidlech. Pro větší nasazení je ideální nějaký vlastní dedikovaný linuxový stroj s odpovídajícím množstvím operační paměti a rychlým diskovým polem.

Během spouštění kontroluje aplikace dostupnost použitých PHP rozšíření a nastavení některých konfiguračních voleb PHP. Zjišťuje nastavení následujících hodnot konfiguračních parametrů jazyka PHP:

- maximální množství paměti, kterou může běžící PHP skript alokovat (memory_limit) je alespoň 82 MiB,
- maximální velikost POST dat (post_max_size) je alespoň 80 MiB,
- maximální velikost nahrávaného souboru (upload_max_filesize) je 64 MiB nebo více.

Reálné množství alokované paměti i velikost ostatních hodnot může být výrazně vyšší nebo i nižší. Záleží na způsobu konfigurace a používání aplikace.

Webový server by měl být zkompilovaný s podporou SSL. Použité certifikáty by měly být vygenerovány nějakou důvěryhodnou certifikační autoritou.

Reálné požadavky klientských aplikací závisí na konkrétním použití. Během jejich spouštění kontroluje aplikace nastavení PHP interpretu.

4.3 Serverová aplikace

4.3.1 Adresářová struktura

V nejvyšším adresáři aplikace se nacházejí soubory: index.php, logdump.php, logclean.php a requests_processor.php. Soubor index.php zpracovává všechny příchozí webové požadavky tak, že inicializuje a spouští příslušné JSON-RPC rozhraní. Soubory logdump.php a logclean.php jsou CLI skripty pro práci s aplikačními logy. Soubor requests_processor.php spouští zpracovatele požadavků.

Adresář config

Obsahuje konfigurační soubor aplikace.

Adresář data

Při prvním použití šifrování atributů nebo hesel vygeneruje aplikace klíče pro symetrické šifrování dat a uloží je do souboru aes_keys.php. Soubor by měl být dobře zálohován. Jeho ztráta znemožní dešifrování zašifrovaných dat z databáze.

Adresář libs

Obsahuje aplikační třídy aplikace. Jejich umístění odpovídá zvolenému jmennému prostoru (v podadresáři Base jsou třídy ve jmenném prostoru *Netsvig\Base* atd.). Díky tomu mohou být dynamicky načítané autoloaderem *Base\Autoloader*. Výjimku tohoto pravidla tvoří adresář 3rdparty, který obsahuje aplikační třídy třetích stran a jejich vlastní autoloadery.

Adresář logs

V případě závažných problémů ukládá aplikace chybové zprávy nebo výjimky do textových souborů v adresáři logs. Na začátku vykonávání skriptu se v adresáři vytvoří prázdný textový soubor s unikátním názvem. Po dokončení skriptu se soubor vymaže. Díky tomu lze detekovat jakékoliv přerušení práce skriptu vzniklé například v důsledku výpadku komunikace s klientem tak, že v adresáři zůstal prázdný textový soubor.

Adresář setupcli

Po základním nastavení aplikace a zprovoznění připojení k databázi je potřeba vytvořit potřebnou datovou strukturu – atributy, kontejnery, role, … K tomuto účelu slouží CLI skripty umístěné v adresáři setupcli.

4.3.2 Konfigurace

Běh serverové aplikace předpokládá nakonfigurovaný databázový server PostgreSQL s vytvořenou databází. Aplikace bude k databázi přistupovat pod nastaveným databázovým uživatelem, který musí mít plná oprávnění ke všem příslušným databázovým tabulkám, funkcím a triggerům.

Konfigurace celé serverové aplikace je uložena v souboru config.php. Nastavuje se zde:

- časová zóna a úroveň logování (0-4),
- připojení k databázi (server, port, název databáze, uživatel, heslo),
- odesílání e-mailů (nastavení připojení k SMTP serveru, textový podpis pro snadnější identifikaci odesílatele, název atributu obsahující e-mailovou adresu identit),
- maximální počet znaků v názvech vytvářených identit.

Správu schémat atributů, rolí, kontejnerů, schvalovacích procesů a systémů nebo vytváření identit s přístupem k administrátorskému JSON-RPC rozhraní aplikace umožňují CLI skripty v adresáři setupcli. U každého skriptu bude uveden konkrétní příklad použití v shellu operačního systému Linux.

Správa schémat atributů

K vytváření, odstraňování nebo modifikaci schémat atributů slouží PHP skript attributedefinition.php, příkaz 4.1 zobrazí nápovědu k jeho používání. Příkaz 4.2 ukazuje vytvoření atributu *telephoneNumber*. Přepínač *-g* určuje název skupiny, *-t* datový typ, *-d* popisek, *-v* validační řetězec, *-f* filtrační řetězec,

```
$ ./attributedefinition.php help
```

Příkazy 4.1: Zobrazení nápovědy příkazu pro správu schémat atributů

```
$ ./attributedefinition.php create telephoneNumber -g core -t cistring -d "Telefonní
číslo" -v "valuescount 1 1 & strlen 1 2500" -f "any ? trim"
```

Příkazy 4.2: Vytváření schématu atributu

Správa rolí

Následující příkazy ukazují vytvoření rolí *osoba* a *superadministrators*. Přepínač *-d* určuje popisek, *-i* prioritu role, *-p* oprávnění, *-s* přiřazený systém. Pro nápovědu lze skript spustit s přepínačem *help*.

```
$ ./role.php create osoba -d "Osoba" -i 100 -s hr -a "attribute=fullName#required=no"
-a "attribute=mail#required=no"
$ ./role.php create superadministrators -d "Super administrátoři" -i 10 \
-p "target_role=osoba#permission_name=CREATE_IDENTITY" \
-p "target_role=student#permission_name=CREATE_IDENTITY" \
-p "target_role=osoba#permission_name=DELETE_IDENTITY#approval_process=sefove" \
-p "target_role=osoba#permission_name=DELEGATE_APPROVAL" \
-p "target_role=osoba#permission_name=DELETE_DELEGATED_APPROVAL" \
-p "target_role=osoba#permission_name=CHANGE_NAME" \
-p "target_role=osoba#permission_name=CHANGE_PASSWORD" \
-p "target_role=osoba#permission_name=CHANGE_ENABLED" \
-p "target_role=osoba#permission_name=CHANGE_USERINTERFACEACCESS" \
-p "target_role=osoba#permission_name=CHANGE_ADMININTERFACEACCESS" \
-p "target_role=osoba#permission_name=MOVE_TO_CONTAINER" \
-p "target_role=osoba#permission_name=ADD_ROLE#resource=role:osoba" \
-p "target_role=osoba#permission_name=ADD_ROLE#resource=role:student" \
-p "target_role=osoba#permission_name=ADD_ROLE#resource=role:ucitel" \
-p "target_role=osoba#permission_name=REMOVE_ROLE#resource=role:osoba" \
-p "target_role=osoba#permission_name=REMOVE_ROLE#resource=role:student" \
-p "target_role=osoba#permission_name=REMOVE_ROLE#resource=role:ucitel" \
-p "target_role=osoba#permission_name=ALTER_ATTRIBUTE#resource=attribute:fullName" \
-p "target_role=osoba#permission_name=READ_ATTRIBUTE#resource=attribute:fullName" \
-p "target_role=osoba#permission_name=READ_ATTRIBUTE#resource=attribute:mail"
```

Příkazy 4.3: Vytváření rolí

Správa kontejnerů

Příkazy 4.4 ukazují způsob spravování kontejnerů. První příkaz zobrazuje vytvoření kontejneru osoby. Přepínač *-d* nastavuje popis, *-r* nastavuje roli a způsob, jakým s ní kontejner pracuje. Druhý příkaz vypisuje existující kontejnery. Pro nápovědu lze skript spustit s přepínačem *help*.

```
$ ./container.php create osoby -d "Osoby" -r "role=osoba#required=yes#default=yes" -r
"role=superadministrators#required=no#default=no"
$ ./container.php show
osoby
 description: Osoby
 roles:
  role name: osoba
    required in all identities: yes
    default for new identities: yes

  role name: superadministrators
    required in all identities: no
    default for new identities: no
```

Příkazy 4.4: Správa kontejnerů

Správa schvalovacích procesů

Ke správě schvalovacích procesů slouží skript approvalprocessdefinition.php. Vytváří, upravuje nebo odstraňuje předpis, na základě kterého se ve vhodných okamžicích spouštějí schvalovací procesy. Práci s ním znázorňují příkazy 4.5. První příkaz vytváří proces s názvem *ucitele*. Přepínač *-l* určuje životnost procesu v sekundách, *-a* nastavuje, zda se mají expirované požadavky schvalovat. Přepínač *-s* přidává schvalovací krok, přičemž *position* určuje jeho umístění. Druhý příkaz vypisuje vytvořené procesy.

```
$ ./approvalprocessdefinition.php create ucitele -l 180 -a no -s
"position=1000#lifetime=60#approveonexpiration=yes#approver=ucitel1" -s
"position=1000#lifetime=60#approveonexpiration=no#approver=ucitel21#approver=ucitel22"
$ ./approvalprocessdefinition.php show
ucitele
 life time (in seconds): 180
 approve on life time expiration: no
 approval steps:
  #1
    life time (in seconds): 60
    approve on life time expiration: yes
    approvers:
        ucitel1
  #2
    life time (in seconds): 60
    approve on life time expiration: no
    approvers:
        ucitel21
        ucitel22
```

Příkazy 4.5: Správa schvalovacích procesů

Správa systémů

Připojení koncových systémů lze nastavovat skriptem system.php. Příkazy 4.6 ukazují vytvoření systému a výpis všech existujících systémů. První příkaz vytváří systém *joomla*. Přepínač *-d* nastavuje popis, *-k* heslo, *-p* určuje schopnost práce systému s hesly, *-e* určuje, zda se při mapování účtů na identity musí rovnat název účtu s názvem identity, *-l* určuje životnost účtu po odmapování (typicky vlivem smazání) identity, *-b* přidává nový mapovaný atribut, *-i* nastavuje, zda má aplikace ignorovat stav, kdy v koncovém systému neexistují požadované účty. Příkaz s parametrem *show* zobrazuje vytvořené systémy. Stejně jako všechny předchozí příkazy, parametr *help* zobrazí nápovědu k použití skriptu.

```
$ ./system.php create joomla -d "Joomla" -k HESLO -p yes -e yes -l 1200 -b
"attribute=fullName#mapping_equality_check=no#idm2system_sync=yes#system2idm_sync=yes#i
dm2system_filter=any ? trim#system2idm_filter=any ?
trim#firstsync_action=replace_system_values" -b
"attribute=telephoneNumber#mapping_equality_check=no#idm2system_sync=yes#system2idm_syn
c=yes#idm2system_filter=any ? trim#system2idm_filter=any ?
trim#firstsync_action=replace_system_values" -i no
$ ./system.php show
joomla
 description: Joomla
 auth key: HIDDEN
 password capability: yes
 no defined random password format, uses identity password
 ignore missing user: no
 new user name filter string:
 mapping user name equality check: yes
 life time after identity unmap (in seconds): 1200
 attribute binds:
  attribute name: fullName
    mapping equality check: no
    idm2system sync: yes
    system2idm sync: yes
    first sync action sync: replace system values
    idm2system filter string: any ? trim
    system2idm filter string: any ? trim

  attribute name: telephoneNumber
    mapping equality check: no
    idm2system sync: yes
    system2idm sync: yes
    first sync action sync: replace system values
    idm2system filter string: any ? trim
    system2idm filter string: any ? trim
```

Příkazy 4.6: Správa systémů

Vytváření identit

Vytváření identit je práce pro administrátorské rozhraní připojené pomocí JSON-RPC. Nicméně nejprve musí nějaká administrátorská identita (uživatel) existovat, aby mohla spravovat další identity. O vytváření identit s přístupem ke všem uživatelským

rozhraním aplikace se stará skript identity.php. Je to relativně omezený nástroj, který by se měl použít jen při prvotním nastavování aplikace. Následující příkazy ukazují, jakým způsobem se s ním pracuje.

```
$ ./identity.php help
Usage: ./identity.php command OPTIONS
Available OPTIONS:
-------------------------------------------------------

For help type: ./identity.php help [command]

Command 'find' IDENTITY NAME:
Command 'createadmin' IDENTITY_NAME:
        -p, --password=<String>
                Identity password

        -c, --container=<String>
                Identity container

        -r, --role
                Identity role; multiple values accepted

        -a, --attribute
                Identity attribute; multiple values accepted
Command 'delete' IDENTITY_NAME:
$ ./identity.php createadmin vasek -c osoby -p Hes456lo -r osoba -r superadministrators
-a "firstName=Václav" -a "lastName=Bohata" -a "mail=vaclav@domena.cz"
$ ./identity.php find vasek
[osoby]/vasek
 id: 1
 admin interface access: YES
 user interface access: YES
 member of roles: superadministrators, osoba
```

Příkazy 4.7: Správa administrátorských identit

Práce s logy

Práci s logy aplikace obstarávají skripty logdump.php a logclean.php. Jsou umístěné v hlavním adresáří aplikace. Oba mají stejnou syntaxi, ale rozdílnou funkci. Skript logdump.php vyhledává logy v databázi a vypisuje je na obrazovku, logclean.php nalezené logy z databáze maže. Práci se skriptem logdump.php ukazují příkazy 4.8.

```
$ ./logdump.php help
Usage: ./logdump.php command OPTIONS
Available OPTIONS:
-------------------------------------------------------

For help type: ./logdump.php help

        -f, --fromtime=<String>
                Show only logs created after date and time, format: YYYY-MM-DD HH:MM:SS

        -t, --totime=<String>
                Show only logs created before date and time, format: YYYY-MM-DD
HH:MM:SS

        -l, --level
                Only if level is ... available options: DEBUG, INFO, WARN, ERROR;
multiple
                levels can be specified

        -s, --loggedsystems
                Include only logged in systems

        -u, --loggedusers
                Include only logged in users
$ ./logdump.php -f "2015-04-22 19:59:30" -t "2015-04-25 23:55:18" -l ERROR -l WARN
```

Příkazy 4.8: Prohlížení logů

Podle potřeby si lze práci s logy zautomatizovat. Příkazy 4.9 v BASHi ukazují, jak si lze z Linuxu nechat zasílat denní logy určité úrovně e-mailem. Využívá se při tom program sendemail. Po úpravě cest lze uvedené příkazy umístit do skriptu, který by se spouštěl jednou denně. Po další lehké úpravě by šlo s použitím skriptu logclean.php odeslané logy vymazat z databáze.

```
$ VCERA="`date -d '-1 day' '+%Y-%m-%d %H:%M:%S'`"
$ DNES="`date '+%Y-%m-%d %H:%M:%S'`"
$ LOGSOUBOR="/tmp/ns_logdump"
$ ./logdump.php -f "$VCERA" -t "$DNES" -l ERROR -l WARN -l INFO > "$LOGSOUBOR"
$ sendemail -f root@server.domena.cz -t vaclav@domena.cz -u "Logy aplikace" -m "Nove
denni logy." -a "$LOGSOUBOR"
```

Příkazy 4.9: Automatizace – odesílání denních logů e-mailem

4.4 CLI klient pro administrátory

Klientská aplikace určená administrátorům potřebuje k provozu PHP 5.4.0 nebo vyšší. Ke komunikaci se serverovým JSON-RPC rozhraním používá v PHP cURL funkce knihovny libcurl. Je do značné míry multiplatformní. Testována byla pod operačním systémem Microsoft Windows (konkrétně Windows 7) a linuxovými distribucemi (konkrétně CentOS 6 a Ubuntu 14.04).

4.4.1 Adresářová struktura

V nejvyšším adresáři se nacházejí soubory: admincli.php, phpwin.bat, pwprompt.bat, startcmd.bat a utf8win.bat. Soubor admincli.php je CLI skript pro komunikaci se serverovým rozhraním aplikace. Pro jeho správnou funkci při zpracování všech textových řetězců je potřeba, aby příkazový řádek byl nastaven na znakovou sadu UTF-8. To v linuxových distribucích není problém, dokonce to bývá výchozí nastavení. Příkazový řádek Windows je ale nutno nastavit. K tomu stačí v příkazovém řádku spustit dávkový soubor utf8win.bat nebo rovnou spustit startcmd.bat – ten spustí příkazový řádek, nastaví znakovou sadu a zobrazí nápovědu skriptu admincli.php. Před zahájením správy uživatelů je potřeba se pomocí admincli.php přihlásit uživatelským jménem a heslem. Není dobré zadávat heslo přímo jako součást příkazu, proto se na něj admincli.php zeptá interaktivně. Aby nebylo zadávané heslo viditelné, vypne se v linuxových systémech zobrazování znaků terminálu. Novější Windows nic takového jednoduše neumí. Proto se používá pomocný dávkový soubor pwprompt.bat, který pro tento účel spustí pomocný příkaz v PowerShell jazyku. Ten bude nahrazovat zadané znaky hvězdičkami, výsledek zadaného hesla bude přečten skriptem admincli.php. Nevýhodou tohoto přístupu může být někdy delší interval prvního spouštění PowerShellu. Dávkový soubor phpwin.bat spouští příslušný PHP interpret, kterému předává všechny zadané argumenty (příklad použití: phpwin.php admincli.php help).

Adresáře GetOptionKit, JsonRPC, Utils

Adresáře GetOptionKit a JsonRPC obsahují zdrojový kód třetích stran, který používá admincli.php pro zpracování argumentů příkazového řádku, respektive ke komunikaci se serverovou aplikací. Utils obsahuje pomocnou statickou třídu pro zpracování řetězců.

Adresář data

Po přihlášení uživatelským heslem vrátí server token, který admincli.php dále používá pro ověřování místo hesla. Jméno přihlášeného uživatele, jeho token a informace o platnosti tokenu se ukládají do adresáře data.

Adresář phpwin

Obsahuje binární soubory, knihovny a konfiguraci interpretu PHP pro Windows.

4.4.2 Konfigurace připojení k serveru

Připojení k serveru se nastavuje přímo v souboru admincli.php. Na začátku souboru jsou dvě proměnné – $admin_url a $ssl_verify_peer. První určuje URL umístění serverového rozhraní. Druhá určuje, zda se při připojování k serveru přes HTTPS má kontrolovat serverový certifikát. Její hodnota by měla být vždy nastavena na true. K tomu je zapotřebí, aby na serveru byl nainstalován certifikát od důvěryhodné certifikační autority. V linuxových distribucích by neměl být problém s důvěrou k takovým certifikátům. Na Windows je ale potřeba nastavit v souboru php.ini v adresáři phpwin konfigurační volbu *curl.cainfo*. Její hodnota musí obsahovat absolutní cestu ke kořenovému certifikátu certifikační autority, která vydala certifikát serveru.

4.4.3 Používání aplikace

Klient nikdy neposílá serveru přímo ověřovací token, ale pouze jeho hash. Součástí hashe je i časové razítko. Je nezbytné, aby se systémový čas na počítači klienta nelišil od času na počítači serveru o více než 10 sekund. To platí i pro další dva klienty. Token má omezenou životnost. Během práce s admincli.php se může stát, že aplikace automaticky token obnoví. Při delší nečinnosti bude potřeba znovu se přihlásit heslem.

Všechny následující ukázky příkazů jsou pro systém Linux. Příkaz 4.10 ukazuje úspěšné přihlášení uživatelským heslem. Neexistuje žádný příkaz pro odhlášení. Po nějaké době přestane přihlašovací token platit.

```
$ ./admincli.php login -u vasek
Enter Password:
Authentication succeeded
```

Příkazy 4.10: Přihlášení heslem k rozhraní pro administrátory

Pro zobrazení nápovědy stačí admincli.php spustit s parametrem *help*. Zobrazenou nápovědu lze specifikovat (*./admincli.php help login, ...*).

Následující ukázky 4.11 zobrazují použití admincli.php při vytváření žádostí na přidání nových identit. První příkaz skončí chybou, protože role *student* vyžaduje atribut *studentClassName*. Přepínač *-c* nastavuje kontejner, *-r* přidává roli, *-a* hodnotu atributu, *-d* popis žádosti, *-w* důvod vytvoření žádosti. Heslo se nastavuje přepínačem *-p*, nezadává se interaktivně jako během přihlašování administrátora. To umožní lepší použitelnost admincli.php v různých skriptech.

```
$ ./admincli.php create_identity_req student016 -p Heslo489 -c studenti -r student -a
"firstName=Oliver" -a "lastName=Drbavý" -a "fullName=Oliver Drbavý" -d "Vytvoření
studentského konta pro nového žáka" -w "Nový žák"
Following error occured:
Error code 444, message: There is a problem with creating a new request, description:
Identity could not be saved because there are some problems with missing required
attributes: Identity 'student016' problem, attribute 'studentClassName' is required by
role 'student' but is missing and no default value exists
$ ./admincli.php create_identity_req student016 -p Heslo489 -c studenti -r student -a
"firstName=Oliver" -a "lastName=Drbavý" -a "fullName=Oliver Drbavý" -a
"studentClassName=1.A" -d "Vytvoření studentského konta pro nového žáka" -w "Nový žák"
Request created successfully
```

Příkazy 4.11: Vytvoření žádosti na přidání identity

Po zpracování dotazu vytvoří serverová aplikace novou identitu a konektory přidají účet do vhodných koncových systémů. Následující příkaz vytvořenou identitu vyhledá a zobrazí její atributy (*-t*) i přidělené systémy (*-m*). Pokud je u nějakého vypisovaného systému „(PASSWORD INCAPABLE)“, pak koncový systém nepodporuje hesla. Text „associated, but not mapped“ znamená, že identita má mít účet v koncovém systému, ale ještě se tak nestalo.

```
$ ./admincli.php identity_search -tm -n student016
 - searching 10 identities, offset 0:

[studenti]/student016
 id: 15
 admin interface access: NO
 user interface access: YES
 member of roles: student, osoba
 attributes:
        activedirectory.ad-uidNumber[INTEGER]:     10016
        core.fullName[CISTRING]:                   Oliver Drbavý
        core.lastName[CISTRING]:                   Drbavý
        core.firstName[CISTRING]:                  Oliver
        student.studentClassName[CISTRING]:        1.A
 associated with systems:
        activedirectory: associated, but not mapped
        hr: associated, but not mapped (PASSWORD INCAPABLE)
```

Příkazy 4.12: Vyhledávání identity

Příkaz 4.13 ukazuje, jak vytvořit žádost na odstranění jedné z hodnot atributu *ad-memberOf*.

```
$ ./admincli.php modify_identity_req student015 -d 'Odebrat AD skupinu Administrators'
-w 'Uz nic nebude spravovat' -a '-ad-memberOf=Administrators'
Request created successfully
```

Příkazy 4.13: Odebrání hodnoty atributu

Díky CLI rozhraní si lze mnoho různých operací značně zjednodušit. Následující příkazy 4.14 předvádí zautomatizování některých akcí pomocí příkazů v linuxovém BASHi. První příkaz povolní všechny uživatele, kteří začínají textem *student*. Druhý příkaz zablokuje uživatele, jejichž celé jméno končí *Bačas*.

```
$ ./admincli.php identity_search -tm -n 'student*' | awk -F/ '/^\[.*\]\/.*/{print $2}'
| while read uzivatel; do echo "Povoluji uzivatele $uzivatel ..."; ./admincli.php
modify_identity_req "$uzivatel" -d "povolit" -w "navrat z dovolene" -s enable; done
Povoluji uzivatele student001 ...
Request created successfully
Povoluji uzivatele student002 ...
Request created successfully
Povoluji uzivatele student015 ...
Request created successfully
Povoluji uzivatele student016 ...
Request created successfully
$ ./admincli.php identity_search -a 'fullName=*Bačas' | awk -F/ '/^\[.*\]\/.*/{print
$2}' | while read uzivatel; do echo "Blokuji uzivatele $uzivatel ..."; ./admincli.php
modify_identity_req "$uzivatel" -d "zakazat" -w "nechci zadne Bacase" -s disable; done
Blokuji uzivatele student001 ...
Request created successfully
Blokuji uzivatele student015 ...
Request created successfully
```

Příkazy 4.14: Použití admincli.php s linuxovými příkazy

4.5 Webové uživatelské rozhraní

Webové rozhraní se připojuje k JSON-RPC serverovému rozhraní pro koncové uživatele pomocí cURL funkcí v PHP. Vyžaduje PHP verze 5.4.0 nebo vyšší. Pro uchovávání přihlašovacích údajů (uživatelské jméno, token) využívá sessions. Bylo testováno pod webovým serverem Apache httpd.

4.5.1 Adresářová struktura

Adresářová struktura vychází z aplikace NETSVig password navržené v rámci bakalářské práce. Používá i některé její třídy. V hlavním adresáři je soubor index.php, který zpracovává všechny uživatelské požadavky.

Adresář config

Obsahuje soubor config.main.php s kompletní konfigurací webového rozhraní.

Adresáře css, image, script

Obsahují soubory kaskádových stylů, obrázky a soubory JavaScriptů.

Adresář includes

V adresáři includes se nacházejí soubory aplikační logiky. Tvoří uživatelské rozhraní a reagují na konkrétní požadavky.

Adresář libs

Obsahuje aplikační třídy aplikace.

Adresář locale

Aplikace s uživatelem komunikuje standardně anglickým jazykem. Soubory s překlady se nacházejí v adresáři locale. Na základě nastavení webového prohlížeče uživatele se aplikace pokusí použít jiný jazyk, pokud pro něj existuje lokalizační soubor.

Adresář logs

Může obsahovat chybové nebo tzv. debug logy webového rozhraní.

4.5.2 Konfigurace připojení k serveru

Webové rozhraní obsahuje jediný konfigurační soubor: config.main.php v adresáři config. V souboru se nastavuje URL serverového JSON-RPC rozhraní pro koncové uživatele, ověřování serverového certifikátu, časová zóna a zapínání debug logu. Ověřování serverového certifikátu pro připojení přes HTTPS by mělo být vždy zapnuté.

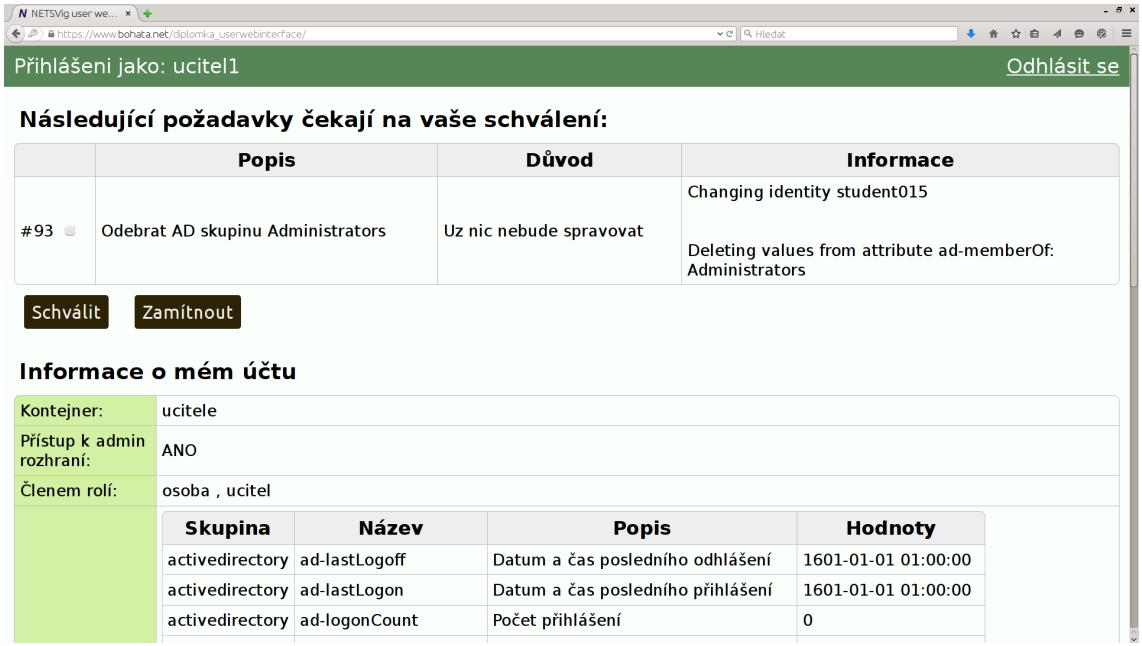
4.5.3 Prostředí webového rozhraní

Aplikace má jednoduché rozvržení. Všechny informace a potřebné akce jsou umístěné na jedné stránce. Po přihlášení do webového rozhraní se uživateli zobrazí informace o jeho identitě. Zobrazí se obsah těch atributů, ke kterým má identita uživatele oprávnění čtení.

Pokud existují nějaké běžící požadavky čekající na schválení přihlášeným uživatelem, zobrazí se jejich seznam. U každého požadavku je zobrazen popis, důvod a bližší informace o prováděných změnách. Každý uživatel má možnost požádat o delegování požadavků, které budou v budoucnu očekávat jeho schválení.

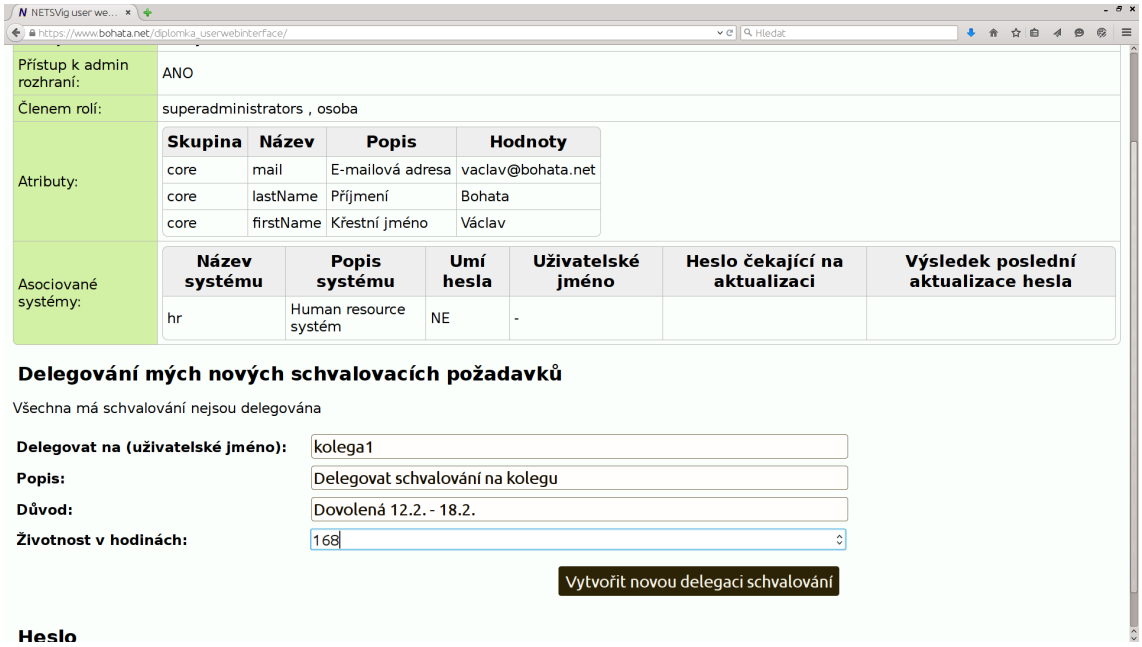
V CLI rozhraní aplikace může administrátor pouze požádat o změnu hesla uživatele (identity). Ve webovém rozhraní si uživatel může změnit heslo rovnou bez vytváření požadavku. Vždy si může změnit heslo identity i heslo účtů ve všech koncových systémech s podporou hesel.

Obrázek 4.1 zobrazuje webové rozhraní přihlášeného uživatele. Jeden požadavek čeká na reakci (schválení nebo zamítnutí) uživatelem *ucitel1*.



Obr. 4.1: Zobrazený požadavek ve webovém uživatelském rozhraní

Obrázek 4.2 ukazuje vytváření delegace schvalování na uživatele *kolega1* kvůli dovolené.



Obr. 4.2: Delegování požadavků ve webovém rozhraní pro uživatele

4.6 Konektor

Pro různé typy koncových systémů jsou zapotřebí různé konektory napsané v různých programovacích jazycích. Ukázkový zde popisovaný konektor je aplikace napsaná v jazyce PHP. Vyžaduje interpret jazyka PHP 5.4.0 a vyšší. Připojuje se k JSON-RPC rozhraní serveru pro koncové systémy. Podpora konkrétních koncových systémů je řešena pomocí pluginů. Každý plugin podporuje nějaký koncový systém. Pluginy jsou nekompatibilní s pluginy použitými v bakalářské práci, část jejich kódu lze ale použít.

Konektor není trvale běžící aplikace, po dokončení aktualizačního cyklu se ukončí. Je vhodné pravidelně ho v určitých časech spouštět. Interval spouštění je závislý na druhu použití.

4.6.1 Adresářová struktura

V hlavním adresáři aplikace se nachází soubor connector.php, který inicializuje a spouští konektor.

Adresář config

Obsahuje konfigurační soubor konektoru config.php.

Adresář libs

Obsahuje všechny třídy aplikace. Kromě podadresáře 3rdparty (třídy třetích stran) musí být všechny třídy umístěné ve správných podadresářích. Podle shod jmenného prostoru s adresářovou strukturou probíhá automatické načítání třídy autoloaderem.

Plugin je představován samostatnou třídou uloženou v adresáři libs/Plugins/Plugin. Aplikace nabízí v základu dva pluginy: univerzální SmartSQL pro správu uživatelů v tabulkách relačních databází a konkrétní vyvinutý ke správě konkrétní databáze v Active Directory.

Adresář logs

Může obsahovat textové soubory s chybovými zprávami vzniklými za běhu aplikace.

4.6.2 Konfigurace konektoru

Konektor se připojuje pouze k jednomu systému. V konfiguračním souboru config.php se nastavuje vybraný plugin pro koncový systém, časová zóna a údaje potřebné pro připojení k serverovému rozhraní aplikace – URL, název systému, klíč a povolení verifikace serverového certifikátu (mělo by být vždy aktivní).

4.6.3 Spouštění konektoru

Na následující ukázce je spouštění konektoru řešeno cyklem v linuxovém terminálu. Je to vhodný způsob pro vyzkoušení činnosti konektoru po jeho nastavení.

```
$ while : ; do ./diplomka_connector_joomla/connector.php; sleep 60; done
Plugin successfully loaded
Successfully connected and authenticated


Processing waiting user name changes ...
No more users to process


Sending list of users to IDM ...
Success


Processing waiting status changes ...
No more users to process


Processing waiting password changes ...
No more users to process


Processing new users to create ...
No more users to process


Processing users to update ...
No more users to process


Processing IDM attributes update ...
No more user needs update


Processing users to delete in our system ...
No more users to delete


Sending "updated" flag...
```

Příkazy 4.15: Spouštění konektoru

5 Závěr

Cílem práce bylo rozšířit aplikaci navrženou v rámci bakalářské práce o API pro vzdálený přístup a o možnost synchronizovat data mezi spravovanými systémy nebo vytvářet automatické akce pro vybrané úlohy správy uživatelských účtů. Po analýze zdrojového kódu se ukázalo, že rozšíření stávající aplikace by mělo za následek přepsání většiny kódu a konfigurace synchronizace by byla zbytečně složitá. Proto byla navržena a vytvořena aplikace nová.

Vytvořená aplikace vychází ze zkušeností s aplikací navrženou v rámci bakalářské práce. Skládá se z několika oddělených celků – serverové aplikace, administrátorské konzole, webového rozhraní pro koncové uživatele a konektoru. Serverová aplikace tvoří hlavní výkonnou část s databází uživatelů a nabízí API pro připojení ostatních celků. Administrátorská konzole je určená pro příkazový řádek a slouží pro spravování uživatelů (identit). Webové rozhraní pro koncové uživatele kromě jiného nabízí hlavně centrální místo, odkud si uživatelé mohou měnit hesla ve spravovaných systémech. Konektory plní funkci prostředníků mezi serverem a koncovými systémy. Pravidelně navazují spojení se serverem, plní jeho příkazy a předávají data ze systémů serveru a naopak.

Centrální správa uživatelů a synchronizace jejich atributů je primární účel vytvořené aplikace. Automatizaci správy uživatelů nabízí na několika úrovních. Dokáže čerpat data vytvořených uživatelů z jednoho systému a distribuovat je do jiných. Automaticky zajišťuje vytváření, odstraňování, aktualizaci i mapování koncových uživatelů. Dokáže kontrolovat hodnoty přenášených atributů a aplikovat na ně transformační pravidla. Seznam pravidel se dá snadno rozšířit vytvořením nových aplikačních tříd. Základní pravidla například dokáží informovat administrátory o překročení hodnoty nějakého spravovaného atributu (například kvóty). Díky mechanismu schvalování správcovských akcí (vytvoření uživatele, úprava atributu, …) lze přenechat správu uživatelů méně kvalifikovaným pracovníkům a pouze schvalovat jejich kroky. Velice mocný nástroj představuje administrátorské rozhraní či program (skript) pro příkazový řádek. To dovolí zkušeným správcům vytvářet různě složité operace podle toho, jak moc ovládají příkazy svého příkazového řádku.

Aplikace je funkční a splňuje požadavky zadání. Budoucí vývoj by se zaměřoval na tvorbu nových validačních a transformačních (filtračních) pravidel a rozšíření jejich interpretu. Dále na vytváření konektorů, přidání možnosti nastavení parametrů hesel do různých systémů a vytvoření webového rozhraní pro administrátory. Podle zkušeností s dlouhodobým provozem by bylo vhodné implementovat mechanismus zpracování binárních dat. Aplikace umí pracovat s binárními hodnotami atributů, nerozumí však jejich významu.

Seznam použité literatury

[1] LÍZNER, Martin. Identity management – centrální správa uživatelských účtů. *Security World.cz | Deník o bezpečnosti pro IT profesionály* [online]. 24. 5. 2010 [cit. 2015-04-07]. Dostupné z: http://securityworld.cz/securityworld/identity-management-centralni-sprava- -uzivatelskych-uctu-2780.

[2] WILLIAMSON, Graham, David YIP, Ilan SHARONI a Kent SPAULDING. Identity management: a primer. Lewisville, TX: Mc Press, 2009. ISBN 978-158347-093-0.

[3] FERRAIOLO, David, D KUHN a Ramaswamy CHANDRAMOULI. *Role-based access control*. Boston: Artech House, 2003, 316 s. ISBN 15-805-3370-1.

[4] *Comprehensive Study of Identity Management Systems* [online]. Lozorno: Evolveum, © 2014-2015 [cit. 2015-04-09]. Dostupné z: https://compare.evolveum.com/

[5] MIKULČÁK, David. CzechIdM is now completely free for your use. In: *BCV CzechIdM blog: BCV solutions s.r.o. company blog* [online]. 17.6.2014 [cit. 2015-04-11]. Dostupné z: http://blog.bcvsolutions.eu/en/czechidm-is-now-completely-free-for-your-use/

[6] MLEJNEK, Jaromír. Pravidla v CzechIdM. In: *BCVlog: Firemní blog BCVsolutions s.r.o.* [online]. 20.3.2012 [cit. 2015-04-11]. Dostupné z: http://blog.bcvsolutions.eu/pravidla-v-czechidm/

[7] Jak na CzechIdM: Úvod pro programátory systému CzechIdM. MATOCHA, Vojtěch. BCV SOLUTIONS S.R.O. *CzechIdM 1.1* [online]. Dec 29 2012 [cit. 2015-04-11]. Dostupné z: http://docs.czechidm.cz/Tutorial/

[8] SEMANČÍK, Radovan. Advanced Hybrid RBAC. In: *MidPoint - Evolveum Confluence* [online]. Jan 16, 2015 [cit. 2015-04-11]. Dostupné z: https://wiki.evolveum.com/display/midPoint/Advanced+Hybrid+RBAC

[9] Groups, Roles and Resources. REVTOVICH, Maria. OPENIAM. *OpenIAM IAM SUITE Ver. 3* [online]. Feb 06, 2015 [cit. 2015-04-12]. Dostupné z: http://wiki.openiam.com/display/IAMSUITEV3/Groups%2C+Roles+and+Resources

[10] Self-Service User Guide. REVTOVICH, Maria. OPENIAM. *OpenIAM IAM SUITE Ver. 3* [online]. Feb 23, 2015 [cit. 2015-04-12]. Dostupné z: http://wiki.openiam.com/display/IAMSUITEV3/Self-Service+User+Guide

[11] Schema, attributes and mapping. PERRONE, Massimiliano. *Apache Syncope* [online]. Oct 24, 2014 [cit. 2015-04-12]. Dostupné z: https://cwiki.apache.org/confluence/display/SYNCOPE/Schema%2C+attributes+and+mapping

[12] Choose workflow engine. BERNHARDT, Jan. *Apache Syncope* [online]. Oct 24, 2014 [cit. 2015-04-12]. Dostupné z: https://cwiki.apache.org/confluence/display/SYNCOPE/Choose+workflow+engine

[13] BOHATA, Václav. W*ebová aplikace pro centralizovanou správu uživatelských účtů*. Liberec, 2012. Bakalářská práce. Technická univerzita v Liberci. Vedoucí práce Mgr. Jiří Vraný, Ph.D.

[14] Transaction Isolation. In: *PostgreSQL 9.4.1 Documentation* [online]. The PostgreSQL Global Development Group, © 1996-2015 [cit. 2015-04-20]. Dostupné z: http://www.postgresql.org/docs/9.4/static/transaction-iso.html

[15] Dependency Injection: kontejner. In: PURCHART, Vašek. *Jak na Dependency Injection* [online]. 12.7.2011 [cit. 2015-04-20]. Dostupné z: http://www.zdrojak.cz/clanky/dependency-injection-kontejner/

[16] Data Mapper. FOWLER, Martin. *Martin Fowler* [online]. [cit. 2015-04-21]. Dostupné z: http://martinfowler.com/eaaCatalog/dataMapper.html

[17] *PHP Manual* [online]. The PHP Documentation Group, © 1997-2015 [cit. 2015-04-22]. Dostupné z: http://php.net/manual/en/

A Obsah přiloženého DVD

Na přiloženém médiu se kromě textu diplomové práce, zdrojových kódů a virtuálního stroje nacházejí schémata nebo diagramy v různých běžných souborových formátech. Některé diagramy jsou navíc k dispozici v nativních formátech příslušných aplikací – Astah (.asta), yEd (.graphml), pgModeler (.dbm).

Virtuální stroj (VirtualBox appliance) obsahuje nainstalovanou linuxovou distribuci Debian s nakonfigurovanou aplikací.

Adresářová struktura:

```
aplikace                        Zdrojové kódy a virtuální stroj
├── admincli                     CLI rozhraní
├── connector                    Konektor
├── server                       Serverová část aplikace
└── userwebinterface             Uživatelské rozhraní
schemata                        Schémata a diagramy
├── databaze                     Databázová struktura
│   ├── diagramy                 Diagramy návrhu databáze
│   └── SQL                      Exportovaná struktura DB
└── tridy                        UML diagramy tříd aplikace
text_prace                      Text práce a použité obrázky
└── pouzite_obrazky              Použité obrázky
    ├── architektura_aplikace     Architektura aplikace
    ├── diagramy_databaze         Diagramy DB
    ├── diagramy_postupy          Popis procesů v UML
    ├── diagramy_tridy            Diagramy tříd
    └── screenshoty               Obrazovky virtuálních strojů
```